WebSphere software

IBM

WebSphere Process Server 6: Business Process Choreographer

BPC Queries

Performance Tuning Methodology

For DB2

V1.1

Jonas Grundler, Rolf Bäurle, Gerhard Pfau
IBM Development Lab Boeblingen, Germany

Abstract

Business Process Choreographer is the component in IBM® WebSphere® Process Server that provides support for business processes and human tasks. This paper establishes a performance tuning methodology that helps you to design faster queries for human tasks and business processes. The intended audiences for this paper are architects and developers that are designing client applications for business processes and human tasks, modelers of business processes and human tasks, as well as performance experts and system administrators that aim to tune the system to improve query performance. The performance tuning methodology we are establishing applies to WPS 6 and WPS 6.1. Most features require WPS 6.0.2.3 or higher.

Note: In WebSphere Process Server 6.2, BPC query tables have been introduced as a first-class concept. See [WPS62QueryTables] for more information.

Table of Contents

Audience

The intended audiences for this paper are architects and developers that are designing client applications for business processes and human tasks, modelers of business processes and human tasks, as well as performance experts and system administrators that aim to tune the system to improve query performance. Readers of this document should have basic skills in:

- WebSphere Process Server
- Database management systems including the structured query language (SQL)
- Business Process Choreographer:
    - human tasks
    - business processes
    - built-in views like TASK and PROCESS_INSTANCE

The performance tuning methodology established by this paper applies to WPS 6 and WPS 6.1. Most features require WPS 6.0.2.3 or higher.

Abbreviations

BPM	Business Process Management
WPS	WebSphere Process Server – IBM's WebSphere based BPM infrastructure
BPC	Business Process Choreographer – the component in WPS providing support for business processes and human tasks.
BPC Views	References to the published views of BPC. Sometimes we also talk about *built-in views* or *built-in BPC views*.
BFM	Business Flow Manager – the BPEL-based process engine in BPC
BFM API	The EJB API of the Business Flow Manager. The BFM query API relates to the "query(…)" method of the BFM API. The BFM queryAll API relates to the "queryAll(…)" method of the BFM API.
HTM	Human Task Manager – the infrastructure in BPC that manages human interactions with services
HTM API	The EJB API of the Human Task Manager. The HTM query API relates to the "query(…)" method of the HTM API. The HTM queryAll API relates to the "queryAll(…)" method of the HTM API.
WID	WebSphere Integration Developer – the development environment for WPS
BPEDB	Business Process Engine Database - Contains business process templates and human task templates, as well as instance data of business processes, human tasks, and related artifacts.
EJB	Enterprise Java Beans

1 Introduction

The performance of queries that retrieve lists of business processes or human tasks is an important factor for the overall performance of a BPM application. This paper establishes a performance tuning methodology that helps you to design faster queries for human tasks and business processes. The intended audiences for this paper are architects and developers that are designing client applications for business processes and human tasks, modelers of business processes and human tasks, as well as performance experts and system administrators that aim to tune the system to improve query performance. All these audiences have to work together to come to an optimum result.

The modelers of the business processes and human tasks have to make sure that they are following the modeling and BPC query best practices, and that they provide the data required by the client developers in an appropriate form. This includes making use of built-in data fields of human tasks like priority, due time, description, type ("business category"), etc. where applicable. Often the built-in data fields are not sufficient, especially when larger amounts of business data have to be presented on task lists or process lists. In these cases modelers of business processes have to make wise use of custom properties, query properties and custom tables to provide the client developers with the data they need. One of the worst things that can happen is if the process and human task designers do not provide sufficient business data in a form accessible by a query. This forces the client developers who wants to show a list of human tasks to do a query first to find out which tasks to display, and then for each task to perform at least one more call to get the required business data.

The client and UI designers rely on what the designers of the business processes and human tasks expose. The client designers have to be performance aware to understand where the capabilities provided to them by the human task and business process developers are sufficient and where restructuring is required before they can develop their client application. They also need to be performance aware because their implementation choices have a large impact on the spectrum of capabilities a performance expert and system administrator has later on to tune the system. Later in the paper we will suggest to define the use of query tables[1] instead of coding your queries using the BPC database views directly, exactly for that reason.

Performance experts and system administrators are the final link in the chain. They get the applications, install them, and configure all components of the Business Process Management System (BPMS). Their job is to ensure that the applications perform as expected by the end users. The options they have in tuning the applications highly depend not only on the available physical resources but also on the application design.

For all people involved in creating high performing applications for business processes and human tasks there are a couple of ground rules that should be followed. Make sure that performance testing is part of application development and that a test environment is used that reflects the production environment as closely as possible. Also make sure

[1] In WebSphere Process Server 6.2, BPC query tables have been introduced as a first-class concept. See [WPS62QueryTables] for more information.

that realistic load testing is performed. If, for example, you are expecting to hit the production system with 10,000 users, each one doing one query per minute then make sure you have simulated exactly this. The performance testing will not only help you better design you application, but also ensures that your team is familiar with the performance tuning techniques required when setting up the production environment.

During performance testing make sure that your performance expert has deep database and SQL performance tuning skills. Assume when tuning your database that you are tuning for both, OLTP and data warehousing scenarios at the same time. If your database expert does not understand the difference then you might want to consult another database expert. Depending on the goals you have for your application you might set up your priorities more towards optimizing for OLTP scenarios, improving business process navigation, or optimizing for data warehousing scenarios, improving query response times. When tuning the database, measure the effects of tuning and re-adjust the database indexes and statistics. Assuming your client developer has chosen to use query tables then in most cases, the standard tuning steps together with query tables will result in good query performance. In few cases, more advanced tuning steps may be required.

The full performance potential of SQL and the database can be leveraged by applying materialized views or custom tables – which are referred to as *query tables* as the general term in this paper.

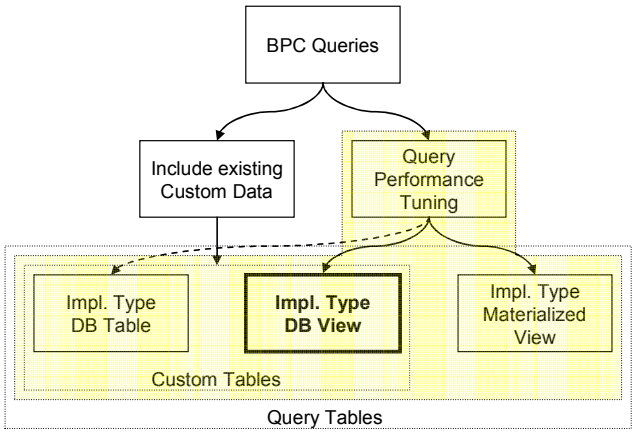


Figure 1: Query Tables Appliances

Query tables are used together with the BPC Query API in two different situations:

- **To include existing, custom data** from tables or views (in BPC queries) which are not part of the BPEDB. In this situation, we talk about "custom tables".
- **To improve the performance** of BPC queries by using materialized views, customized SQL, or custom tables which are used together with views in the BPEDB. In this situation we often use the term *query tables*.

This paper concentrates on how to use query tables for performance tuning, with focus on query tables of implementation type DB View. How to include custom data is described in [CustomTables]; the materialized view mechanism is described in [MatViews] and [CustomTables].

2 Motivating example

The following chart shows performance improvements on DB2 UDB V8 that have been achieved in a lab setup, applying tuning methodologies described in this paper:

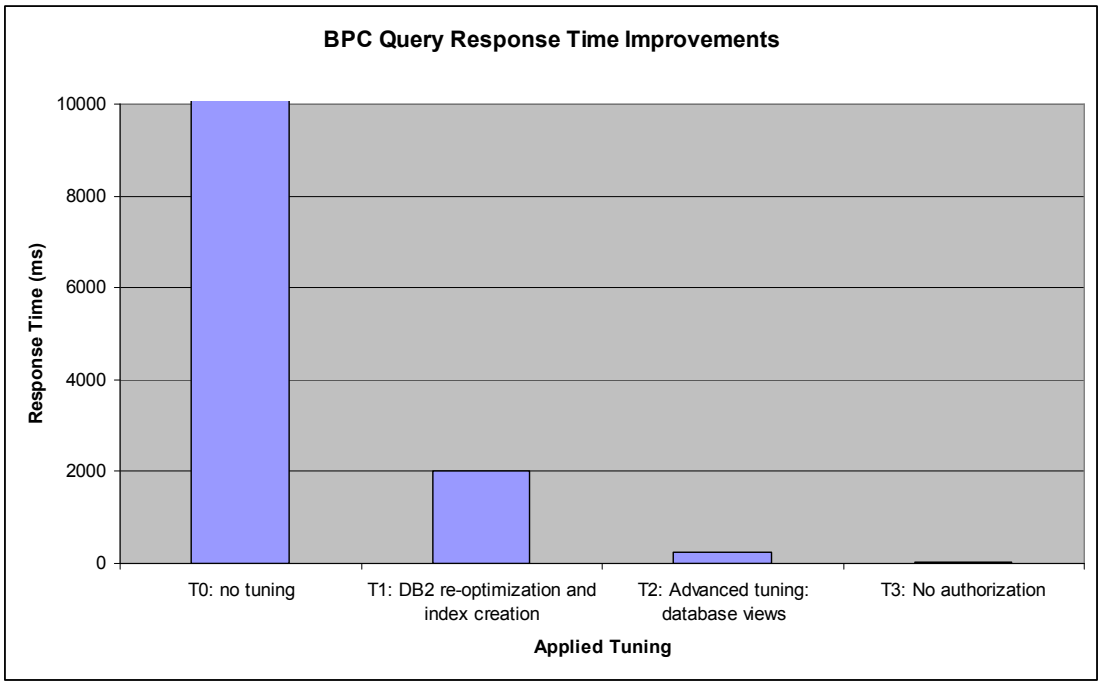


Figure 2: Performance Tuning - Sample Improvements

For the measurements in the figure above, the Business Process Choreographer database (BPEDB) has been loaded with 100,000 business process instances. Each business process instance has two inline human tasks, resulting in 200,000 human tasks in the system in total. All human tasks are in state *ready*, and thus may appear on the task list of business users working with the system. In order to achieve optimum performance, group work items are used for authorization. That is, people assignments for the human tasks have been defined using the verb "Group" for the potential owners of the task.

In order to be able to present business data on a task list, the process has been defined to include 10 query properties. With 100,000 business processes in the system, this results in an overall number of 1 million query properties in the system.

In order to simulate the retrieval of a task list representing the tasks a person is eligible to work on, a BPC API query has is used. The query retrieves all tasks a certain user can see. For each task the task identifier (TKIID) is retrieved including 10 query properties that are bound to the corresponding process instance. In T0-T2 authorization is used (BPC query API query(…)). In T3 no Authorization is used (BPC query API queryAll(…)).

In the test setup we chose, the database (DB2 UDB) and WPS are on different physical machines of the same type: 3GHz single CPU, 4GB memory. The database uses about 1GB of memory. The type 4 JDBC driver is used to connect WPS to the database.

The results shown in the figure above have been achieved with single threaded task list queries using the following techniques:

T0 (No tuning): At least 10 seconds response time. Typically, response times vary a lot if the system is not tuned.

T1 (Best practice):

- Applying DB2 UDB re-optimization reduces the response time to 2 seconds.
- Database views have been created for the query, and registered with BPC using the custom table mechanism (that is, implementation type DB views are used here). Note that the database view uses the same SQL in its definition as the SQL that is generated by BPC[2]

T2 (Advanced tuning): Using query table (implementation type DB view) SQL optimizations (no redundant/physical data in the custom table) and index creation (which is required due to changed SQL) results in approximately 250ms response time.

T3 (No authorization): Removing authorization decreases the response time below 25ms.

For more details on the different tuning steps and the artifacts used please refer to Appendix C.

[2] A join with the WORK_ITEM view is not done here. At runtime, BPC automatically joins the WORK_ITEM view if the BPC query API query() is used and if needed.

3 Query tables

BPC queries (or queries for short) reference built-in BPC views which are published as part of the WPS documentation. Examples for these views are TASK, PROCESS_INSTANCE, WORK_ITEM, or QUERY_PROPERTY (for a complete list see [BPCVIEWS]):

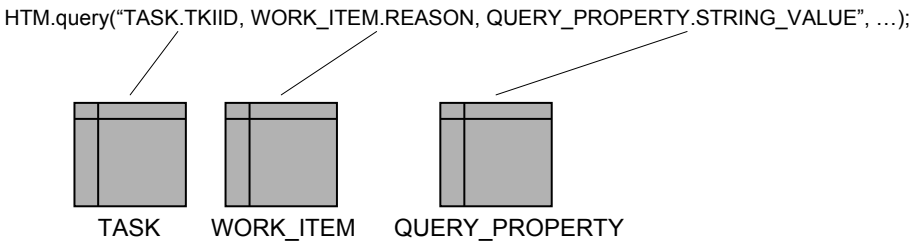


Figure 3: BPC query referencing BPC views

As mentioned in the introduction, query tables[3] can be used for performance tuning. Performance is the main driver for using query tables. The possibility to define an abstraction (identified through a name) from the actual built-in BPC views used (and how they are joined together) which serves a particular set of queries, is a second important aspect. In the sample below, the BPC query references a query table with name MY_TABLE:

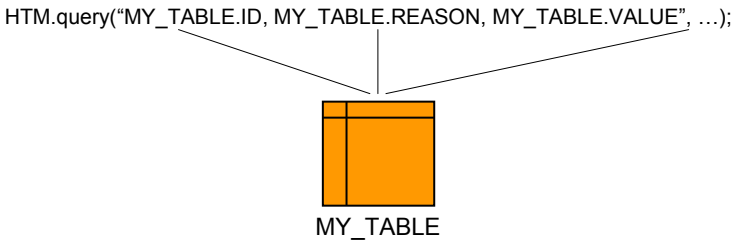


Figure 4: BPC query referencing a query table

In WPS 6.1, a query table gives you the possibility to:

- Define a view on the database level that aggregates all information which you need within your BPC API queries.
- Include custom data into your BPC client application.
- Optimize the SQL which aggregates information on the database layer, using the full power of SQL which may be specific to your database.
- Speed up your queries by adding redundant, but easily accessible data using custom tables or materialized views – without changing your application code.

Further advantages of using query tables are:

- Simplicity: Queries that reference query tables by name, usually reference a single query table only. Also, the where-clause usually is simpler, because part of the where condition is already contained within the query table itself.

[3] In WebSphere Process Server 6.2, BPC query tables have been introduced as a first-class concept. See [WPS62QueryTables] for more information.

- Readability: The name of a query table can be expressive and customized by the application developers together with the database administrators.

The following table lists the WPS versions and query table capabilities along with their specific implementations:

Table 1: Query tables: supporting WPS versions and implementations

Since WPS version	Query table implementations
6.0.2.1	Materialized views[4] on DB2 UDB
6.0.2.2	Materialized views on DB2 for z/OS and Oracle
6.0.2.3	Custom tables on all supported databases
6.1.0.0	Named materialized[5] views on DB2 LUW, DB2 zOS, Oracle

As described in the introduction, there is a variety of techniques available for performance tuning BPC queries with query tables:

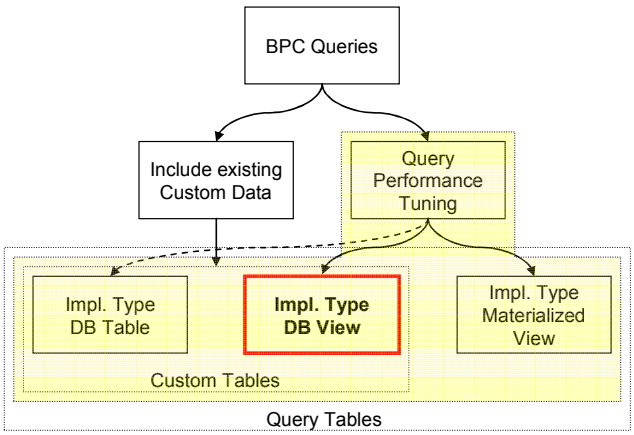


Figure 5: Query Tables Appliances (focus on CT DB views)

Implementation type DB view: In most cases, using implementation type DB view should be sufficient in order to get good query response times.

Implementation type materialized view: Materialized Views refer to the "materialized view mechanism" which is provided by BPC since WPS 6.0.2. Materialized views use a database caching mechanism which can be applied to some scenarios without any big effort, if query performance problems exist.

Implementation type DB table: This type is referring to a custom table which points to a database *table*. Using a physical table for performance tuning BPC queries, should be the last choice, as complex dependencies and a high maintenance effort is introduced (which also requires a deep technical expertise of BPC).

A detailed description about when to use which technique is described in the next chapter.

[4] Note that the materialized view mechanism in WPS 6.0.2.x uses a specific mapping of queries to materialized views, which has some limitations. See [MatViews] for more information.

[5] Documented in [CustomTables]

4 Methodology

To improve the response times of queries, several performance tuning steps can be applied. Figure 6 shows the recommended methodology for applying BPC tuning steps. Tuning and performance tuning steps are done on the test system, as there can be several iterations of the performance tuning methodology which may result in database artifacts (such as database tables or views) to be created and changed. Once the final set of best practices and artifacts has been found, these artifacts need to be applied to the production system together with the application:

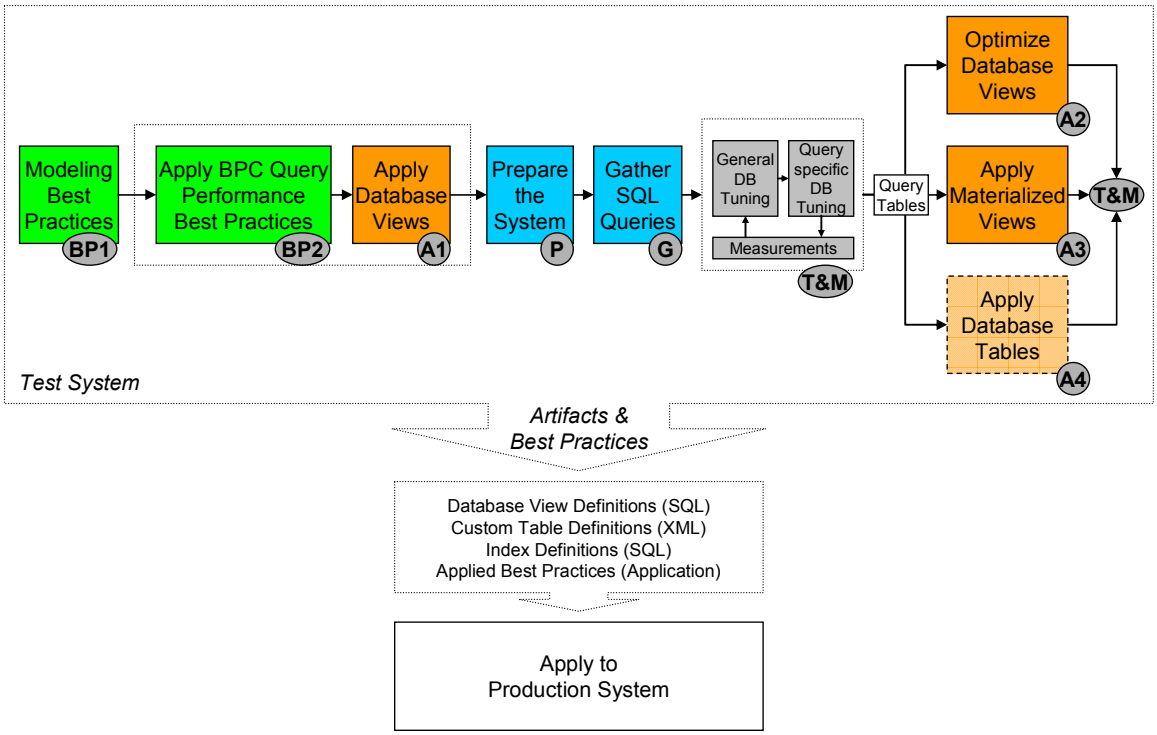


Figure 6: BPC Query Performance Tuning Methodology

Tuning Steps: Applying "modeling best practices" is a standard tuning step, which should be considered in every development process; (BP1) is focused on modeling best practices which have impact on query and overall system performance. "BPC query performance best practices (BP2)" are focused on specific best practices for query performance. Especially the choice of using "Query Tables of implementation type database views (A1)" has impact on these best practices; therefore, make sure that applied BP2 are aligned with the advanced tuning step chosen.

Strongly consider tuning step (A1) "Apply Database Views". Tuning step (A1) gives you the flexibility to further optimize the SQL which is used to access the data in the BPC schema. Also, later on, you have the flexibility to move to one of the more advanced mechanisms like materialized views or (query tables of implementation type) DB tables without changing any application code. This is because the name of the view you are using in your queries does not change. Alternatively, you can proceed without having implemented task (A1) and performance tune the system with the default implementation that makes no use of any custom table or materialized view. If you do so the improvement potential is limited though.

Preparing the system (P) is mandatory for the assessment of the performance of your queries. The preparation phase includes the definition of the test cases and the loading of data into the test system. Both the test case and the amount of data loaded into the system have to be representative for the workload expected at production time.

There are eight tuning steps (BP1 and BP2, A1-A4, P, G) which are performed on WPS related components and the database, and one tuning step which is performed on the database only (T&M). The diagram is read from the left to the right: start with tasks (BP1), (BP2), (A1), (P), (G), and then execute (T&M). If the performance of your queries does not meet your requirements, evaluate which step should be taken next, based on your requirements. Often a tuning step has the potential to further improve the BPC query response time by a factor of 10 or more.

Maintenance Effort: Each tuning step requires a certain amount of time for the implementation and for further maintenance. (BP1) is implementation only, and therefore an on-going effort while the application evolves. Tasks (P), (G), (BP2), and (T&M) are mandatory, and may take two weeks or longer. Task (A1) is simply a rewriting of the SQL (into a database view), an XML file creation, and few changes to the application code. Plan about 2-3 days for this task. Tasks (A2), (A3), and (A4) may take a week or longer.

Implementation: The recommended way of tuning BPC queries is, to apply one tuning step after the other:

- Tuning steps (BP1) and (BP2) should be revisited during application development and after performance tests and measurements.
- Tuning step (A1) and the preparation step (P) must be executed before any other tuning step is applied. As said above, tuning step (A1) could be omitted, but it is strongly recommended not to do so.
- Tuning steps (G) and (BP) must be applied before making any assumption about the resulting query performance.
- The repetition of general database tuning and query specific tuning (T&M) is mandatory for the evaluation of the improvements made in the tuning step before.
- Tuning steps (A2), (A3), and (A4) are usually implemented mutually exclusive, that is: if in your particular scenario tuning step (A2) does not result in the expected performance improvements, try task (A3), and if that did not suffice either go with (A4).

Please note that different queries might require different methodologies. Therefore it is normal that on a single WPS server (or cluster), different tuning techniques (advanced tuning steps (A3) and (A4)) are used for different queries in order to satisfy the performance requirements.

4.1 Tuning Step A1

Tuning step (A1) reflects what you have to do in order to use query tables *by name*. We recommend doing this tuning step, and using your own views or tables in BPC queries instead of directly using the built-in BPC views shipped by WPS. Please note that when defining your views you will reference BPC's built-in views.

The following query provides an example:

Table 2: Sample query

```
// htm is the HumanTaskManager EJB interface
htm.query(

// select clause
"TASK.TKIID, TASK.STATE, QUERY_PROPERTY.STRING_VALUE, PROCESS_INSTANCE.NAME",


// where clause
"TASK.STATE=TASK.STATE.STATE_READY AND QUERY_PROPERTY.NAME='customer'",
…
```

Using query tables, the query can look like (requires WPS 6.0.2.3 or higher):

Table 3: Sample query using query tables by name

```
// htm is the HumanTaskManager EJB interface
htm.query(

// select clause
"MY_TASKS.TKIID, MY_TASKS.STATE, MY_TASKS.CUSTOMER, MY_TASKS.PI_NAME",


// where clause (defined in the database view)
null,
…
```

4.2 Tuning Steps A2 – A4

In most scenarios, we expect that the implementation of the standard tuning steps and advanced tuning steps (A1) and (A2) is sufficient. However, if further tuning is required, take into account that the level of the Quality of Service, disk space needs and other properties should be considered when applying tuning steps (A3) or (A4). For instance, if the data which is returned must be current, materialized views are not the best choice because they contain cached data, and depending on their refresh interval, this data might be too inaccurate for a specific query. For more details on materialized views please refer to the white paper *Performance Tuning of Human Workflows Using Materialized Views* [MatViews].

The following table lists the various attributes of applied tuning techniques (advanced tuning steps):

Table 4: Tuning steps attributes comparison

Task #	**Additional Disk Space Costs**	**Uses Cached Data**	**Navigation Perf. Impact**
(default implementation)	No	No	no
A1 and A2 (database views)	No	No	no
A3 (materialized views)	yes	Yes	no
A4 (database tables)	yes	No	yes

4.2.1 Tuning step A2

This option makes use of SQL features which help the database management system to choose a better access plan when retrieving data. We will describe how to use scalar fullselects in order to speed up BPC queries which contain custom properties, query properties or attributes. Additionally, you can use any SQL technique in order to speed up your BPC queries. This tuning step is explained in detail in section A2: Optimize query tables of implementation type DB Views.

4.2.2 Tuning step A3

Materialized views are a well-known database concept for caching SQL query results. WPS supports the usage of materialized views that are periodically updated when query requests come in, instead of doing automatic updates when the information in the base tables change during business process navigation. As a consequence this technique speeds up BPC queries without negatively impacting BPC navigation performance. A detailed description of tuning step (A3) can be found in section A3: Apply Materialized Views.

4.2.3 Tuning step A4

Custom tables are a methodology which allows you to reference additional "tables or views" within the BPC query API. Initially this was introduced to allow you to join BPC data with custom data from other sources. You can also use custom tables to prepare data of business processes and human tasks in a form that is optimized for particular task list queries. Different technologies can be used in order to maintain the data in your custom tables. More details on this tuning step can be found in section A4: Apply query tables of implementation type DB table

5 Implementation

This chapter describes in detail how to implement the various tuning steps introduced in the previous chapter.

5.1 BP1: Modeling Best Practices

Modeling business processes and human tasks includes a variety of artifacts. Some of them require special attention when modeling applications for production scenarios. Examples for such artifacts are the following

- first-class attributes on BPEL activities and human tasks
- information retrieval (single query vs. multiple queries e.g.)
- people assignment and roles (editor, potential owner, administrator)
- transactional boundaries
- business relevance settings on activities
- synchronous versus asynchronous service invocation
- other

Although most of the settings mentioned above do not immediately have impact on query performance, large-scale scenarios show that "efficient modeling" has positive impact not only on process navigation ("throughput") performance, but also on query performance. Best practices in this chapter relate to long-running business processes only.

First-class attributes on BPEL activities and Human Tasks*[6]: To reduce complexity and to improve performance use first-class attributes (e.g. the field 'priority' on human tasks) instead of using custom properties.

Looping and information retrieval: When retrieving information for, for example, a list of human tasks then an anti-pattern is to retrieve one part of the information with the BPC query API, and then execute additional queries (or a BPC API call like htm.getTask(…)) for each entry that has been returned by the previous query. To reduce the load on the system, and to improve query performance, try to retrieve all data with a single BPC query or with a very limited number of BPC queries.

People resolution*: Use group work items instead of individual work items (e.g., use the people assignment criterion "Group" instead of "Group Members"). Group work items are very effective if a group of users must be assigned (e.g. to a human task).

Roles*: On human tasks and business processes, different roles can be modeled (e.g. potential owner, reader, editor). Only specify a people assignment criterion for those roles that you really need. For instance, if a group of potential owners is defined on a human task, this group automatically gets read authority on the human task by definition; no extra modeling is needed.

Transactional boundaries*: At transactional boundaries, BPC may decide to store additional data in order to support certain scenarios (which are not covered and

[6] Best practices marked with "*" have positive impact on the number of entries in the BPC tables, and therefore have positive impact on query performance.

explained here). Limit transactional boundaries by setting "participates" on BPEL activities if applicable.

Business relevance flags on business processes*: Business process activities can be marked "business relevant". Then, those activities are persisted and can be queried by BPC queries. In case that those activities are not needed to be queried by custom applications, uncheck the business relevance flag. BPC can then often decide not to write those activities to the database, which improves performance.

Service invocations*: Business processes often invoke ("Invoke" activities) Web services in order to execute business logic. Often, these service calls can be done synchronously. The footprint of synchronous service invocations compared to asynchronous service invocations is much smaller, because synchronous service invocations run within a single transaction, and therefore no intermediate state needs to be written to the database. Note that there are multiple settings on the SCA assembly editor which influence whether a service call is made synchronously or asynchronously:

- On the business process's service reference, set "Suspend transaction" to "False".
- On the service's interface, set "Join transaction" to "True".
- On the service's interface, set "Preferred interaction style" to "Synchronous".
- On the service's implementation, set "Transaction" to "Global" (if applicable).

5.2 BP2: Apply BPC Query Performance Best Practices

BPC query performance best practices described in this section apply to both BPC API queries and to SQL queries (as used if tuning step (A1) or JDBC is used). These best practices are derived from large-scale WPS installations.

Omit authorization: For queries where your application does not require the built-in authorization of BPC, use the queryAll(…) API instead of the query(…) API. Note that the query(…) API always adds an implicit join with the WORK_ITEM view, to ensure proper authorization, while queryAll(…) relies on static, J2EE role based authorization. For more details please refer to the JavaDoc in the InfoCenter (see [BFMAPI] and [HTMAPI]).

Where-clause conditions: Narrow the result set with all the knowledge you have about the objects you want to retrieve. For instance, use "TASK.KIND=TASK.KIND.KIND_PARTICIPATING" if you know that you're only interested in "participating human tasks" (human tasks that are modeled in the BPEL process model). Don't be anxious to use too long query expressions but rather be specific. The more precisely you are telling the database what you want to retrieve (and what not), the better is usually the performance of your query.

Threshold: Using a threshold in order to limit the number of returned records is always a good idea and improves BPC query response times.

Join conditions: Avoid join conditions with an "OR" clause. If possible, use "UNION" statements instead. This applies to the database view creation and to JDBC queries only.

Scalar fullselects: Various techniques are available on database systems in order to speed up SQL queries. Some techniques are related to SQL. Scalar fullselects can be used to give the database optimizers a better picture of what the relationship between the

tables in the database is. In this document, we make heavy use of the SQL technique called "scalar fullselects" (see Appendix D).

Join with WORK_ITEM view: If the HTM query API or the BFM query API query() is used, selected views (or custom tables) are joined with the WORK_ITEM view (which holds authorization information)[7]. As the WORK_ITEM view typically has many entries it is worth to be specific about which entries you are interested in. The following conditions should be added to the where-clause in order to provide additional hints to the database:

- If the primary view[8] is the TASK view, add "…AND WORK_ITEM.OBJECT_TYPE=WORK_ITEM.OBJECT_TYPE. OBJECT_TYPE_TASK_INSTANCE AND…"
- If the primary view is the PROCESS_INSTANCE view, add "…AND WORK_ITEM.OBJECT_TYPE=WORK_ITEM.OBJECT_TYPE. OBJECT_TYPE_PROCESS_INSTANCE AND…"

5.3 A1: Apply Query Tables of implementation type DB Views

With WPS version 6.0.2.3 a concept called *Custom Tables* has been introduced in BPC to allow the inclusion of custom database tables or database views when using the BPC query APIs. In this document, custom tables which point to database views are called "query tables of implementation type database views" (as a variation of custom tables).

Custom tables are declared and configured as part of the BPC container settings and after that they can be used in BPC queries. The custom tables are co-located with the BPC tables in the same database. They are often used to include human task related or business process related business data that is not managed by BPC. For more details on custom tables please have a look at [custom tables].

Besides the scenario where custom data is included, custom tables can also be used to improve of query performance. The way described in this section uses the "custom tables" concept together with a clever definition of "database views" in order to speed up queries.

Consider a query which returns the Task ID (TKIID) plus two Query Properties that have been set on the related process instance (so, we're talking about an inline to-do task). The standard query with the BPC API would be:

```
htm.query("TASK.TKIID, QUERY_PROPERTY1.STRING_VALUE, QUERY_PROPERTY2.STRING_VALUE",
"TASK.KIND=TASK.KIND.KIND_PARTICIPATING AND QUERY_PROPERTY1.NAME='Name' AND
QUERY_PROPERTY2.NAME='Street",  null, new Integer(200), null);
```

The resulting SELECT statement contains a join condition between the TASK view, the WORK_ITEM view and two times the QUERY_PROPERTY view.

Most databases have difficulties with this kind of complexity (the referenced views are also complex queries). As well, what we're really looking for is: "give me all human tasks which belong to me, and enrich them with the custom properties (or query properties) of

[7] Typically only one view *V* is joined with the WORK_ITEM view, other views being accessed in the same query are joined with *V*.

[8] Refer to chapter 5.3 for a description of "primary view" and "attached view".

the related process instance. In case that the custom property (or query property) does not exist, return NULL". In technical terms, we want to have a de-normalized view on the list containing TASK.TKIID, QUERY_PROPERTY (customer), and QUERY_PROPERTY (address).

A better way to get this kind of information from the database is to use "scalar fullselects". This SQL technique is available on DB2 and Oracle.

Notes:

- Before reading the example provided in this chapter, make sure that you have a good understanding of scalar fullselects (see Appendix D).
- Aggregation of information between BPC views is done via SQL joins. See Appendix E for rules of how to join BPC views together (using scalar fullselects).
- Examples for task and process lists are provided in Appendix F.
- Authorization: If the BFM query API query() or the HTM query API query() is used, custom tables (therefore, also query tables of implementation type DB View) are joined at runtime with the WORK_ITEM view. If the BFM queryAll(…) API or the HTM queryAll(…) API is used, no join with the WORK_ITEM view is performed at runtime, which may improve query response times.

5.3.1 Primary views and attached views

Typically, a task or process list contains "human task instances" or "process instances". With SQL, you have the possibility to process arbitrary joins between tables or views. In the context of BPC client applications, arbitrary joins do almost never make sense – e.g. you wouldn't like to get the Cartesian product between the TASK view and the QUERY_PROPERTY view. Rather, you'd like to get a list of tasks along with its corresponding entries in the QUERY_PROPERTY view (if existent).

In order to make this relationship more prominent (also see Appendix D), we call the view which determines the main object of the list to be retrieved the *primary view*. BPC views which just act as information provider (such as task properties) are called *attached views*. The following picture visualizes this idea using two examples:

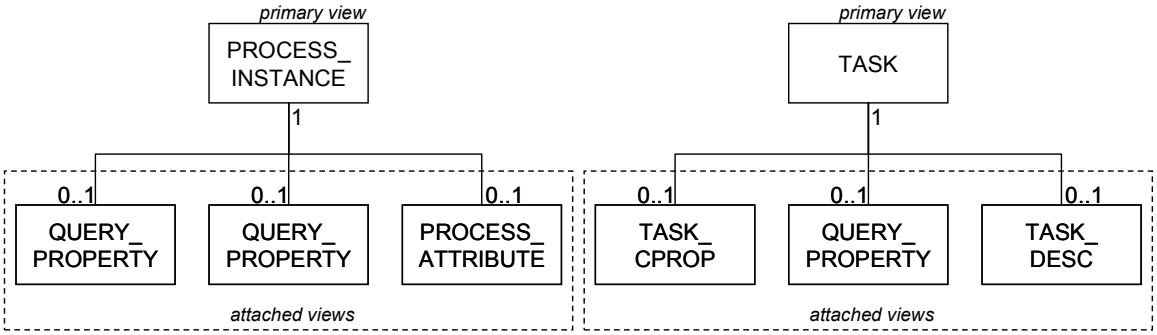


Figure 7: Examples: Primary and attached views

When defining database views then the SQL for a database view is written such that the select statement goes against the primary view (maybe also against an attached view, joined with the primary view in case of a filter criterion on the attached view – see F.3),

and the information from the attached views are added using scalar fullselects (see Appendix D). The primary views and the attached views must be connected (joined); this is done using join conditions. These join conditions are described in Appendix E.

5.3.2 Applying Query Tables of implementation type DB Views Example

This example shows how to apply query tables of implementation type DB Views in order to improve the performance of the query introduced in the section before. Here is the query again, for your convenience:

```
htm.query("TASK.TKIID, QUERY_PROPERTY1.STRING_VALUE, QUERY_PROPERTY2.STRING_VALUE",
"TASK.KIND=TASK.KIND.KIND_PARTICIPATING AND QUERY_PROPERTY1.NAME='Name' AND
QUERY_PROPERTY2.NAME='Street",  null, new Integer(200), null);
```

Please follow the steps below in order to install and use the custom table of this chapter. Keep in mind that you need a business process which defines the related query properties; otherwise "null values" are returned.

Step1: Follow the steps described in chapter 11 of [custom tables] – without creating the physical database table (skip chapter 11.1). Note that reading chapter 11 of [customTables] is a must in order to understand the text of this chapter. A custom table defined in BPC which is accessible by the BPC query API through name "CUSTOM_DATA" should be configured on BPC before proceeding to the next step. The following structure of the custom table is assumed:

- ID (TYPE_ID)
- NAME (TYPE_STRING)
- STREET (TYPE_STRING)
- ZIP (DECIMAL)
- CITY (TYPE_STRING)

Step2: On the database, create the view "CUSTOM_DATA" with the following view definition:

```
CREATE view CUSTOM_DATA
(TKIID, NAME, STREET, ZIP, CITY) AS
SELECT TA.TKIID ,
(SELECT STRING_VALUE FROM QUERY_PROPERTY WHERE TA.CONTAINMENT_CTX_ID=PIID AND
NAME='Name' ),
(SELECT STRING_VALUE FROM QUERY_PROPERTY WHERE TA.CONTAINMENT_CTX_ID=PIID AND
NAME='Street' ),
(SELECT NUMBER_VALUE FROM QUERY_PROPERTY WHERE TA.CONTAINMENT_CTX_ID=PIID AND
NAME='Zip' ) ,
(SELECT STRING_VALUE FROM QUERY_PROPERTY WHERE TA.CONTAINMENT_CTX_ID=PIID AND
NAME='City' )
FROM TASK TA;
```

Create file "customData.sql" and paste the contents of the table above into it.

On DB2 UDB:

Log into BPEDB: `db2 connect to BPEDB`

Create view CUSTOM_DATA: `db2 -tf customData.sql`

Step3: Test in the command line if the view is working:

On DB2 UDB:

Log into BPEDB: `db2 connect to BPEDB`

Select some sample rows: `db2 "select * from CUSTOM_DATA fetch first 10 rows only with ur"`

Step4: Include the custom table into your application:

The following code snippet shows how to query custom table "CUSTOM_DATA" in your application:

```
import com.ibm.task.api.*;
...
InitialContext initialContext = new InitialContext();

// lookup the EJB home interface
Object object = initialContext.lookup("com/ibm/task/api/HumanTaskManager");
HumanTaskManagerHome htmHome = (HumanTaskManagerHome)
javax.rmi.PortableRemoteObject.narrow(object, HumanTaskManagerHome.class);

// get the remote interface
HumanTaskManager htm = htmHome.create();

// query all tasks that the current user is allowed to see
QueryResultSet resultSet = htm.query(
// select clause - what do we want to get?
       "CUSTOM_DATA.ID, CUSTOM_DATA.NAME, CUSTOM_DATA.STREET, CUSTOM_DATA.ZIP,
CUSTOM_DATA.CITY",
// where clause (no where clause in this sample)
       null,
// order clause (no order-by clause in this sample)
       null,
// threshold - return first 200 entries only
       new Integer(200),
// no timezone specified
       (TimeZone)null
);

// loop over results in the result set
while( resultSet.next() )
{
// print out selected columns,
System.out.println( "Task ID (TKIID) = " +
resultSet.getOID(1) );
System.out.println( "Customer name = " + resultSet.getString(2) );
}
```

5.4 P: Prepare the system

The preparation step applies to the test environment used for benchmarking and measurements.

To prepare the system, load the expected amount of business processes and human tasks that you will have in your production environment. Please use applications, business processes, human tasks, and configuration settings etc. that are characteristic for the scenario you are going to have in production. Please be as exact as possible, as small differences can have a big effect. For instance, changes in the staff resolution, by using individual work items vs. group work items can have a huge impact on query performance as well as on runtime navigation performance.

If the system does not reflect the production environment, especially pieces that affect the database and its contents, response time measurements will be no good indicator for what you are going to experience in production. This is particularly true for query response times.

5.5 T&M: DB Tuning and Measurements

This tuning step contains tuning tips related to database tuning and measurement methodologies.

5.5.1 Database Tuning

Chapter 5.5.1.1 gives some basic guidance for general database tuning and index creation. Note that additional indexes improve query performance, while having impact on insert, update, and delete performance on the database. You may want to remove some indexes BPC creates for you if your database performance tools suggest that they are not used in your particular configuration. Please be careful in doing so, as some indexes are needed for deadlock prevention. In case you see deadlocks please add the previously removed indexes again.

5.5.1.1 Top parameters and tuning for DB2 UDB

Query re-optimization: Apply the technote [Improving the performance of complex BPC queries on DB2] to allow DB2 UDB to do the proper recalculation of data access paths.

Disk layout: Ideally you are using a fast disk sub system that provides good latency and throughput. If this is not available you have to use single disks. Distribute logs and table spaces across as many physical disks as you have available; a number of 8 disks is a least minimum for most customers.

Parameters: DB2 UDB offers a multitude of parameters, of which many affect performance. Running the DB2 UDB configuration advisor helps in the initial tuning of these parameters. There are a few parameters that need special attention. Here are our tuning suggestions:

- Give the DB2 UDB bufferpool that is used by the BPEDB (normally IBMDefaultBP) enough memory. In our tests we have set the bufferpool to at least 0.5 GB of memory. For 4K pages, this is 125000 pages.

- Increase the default size of the lock list (LOCKLIST). If you are running out of locks, which is the case when concurrency increases and your lock list is too small then the database may switch to table level locking, instead of row level locking, which has dramatically negative effects on performance and often even leads to deadlocks. In our tests we were working with a lock list size of at least 1000 pages.
- Make sure that you have enough IO cleaners and IO servers. If you are using physical disks then allow for one IO cleaner and IO server per disk (NUM_IOSERVERS and NUM_IOCLEANERS).

Statistics: Database statistics are used by the database to optimize the access path when accessing data. Updating statistics makes no sense as long as no data is in the database; actually it even often hurts performance. Therefore, after the database is loaded with its contents, as well as in between the prepare step (P), run statistics on tables and indexes with distribution. A sample script which can be executed after customization is contained in Appendix A. Keep in mind that the database statistics should be updated before and after database tuning steps, and also regularly on the production system. In order to make DB2 use the new statistics, either restart the database or simply perform the following statement in the DB2 command line:

```
db2 FLUSH PACKAGE CACHE DYNAMIC
```

Note that flushing the package cache or restarting the database may have a temporary performance hit, because dynamic statements need to be compiled again.

SQL Query Access Path Optimization: In step (G), which is further detailed below, the SQL statements which are created on the basis of BPC queries have been collected. Put all statements in a single file "queries.sql". The statements must be separated by a <Return> and a semicolon at the end. Example:

```
SELECT * FROM …. WITH UR;
SELECT TKIID FROM TASK….WITH UR;
```

Then, in the DB2 command line, execute the following command[9]:

```
db2advis -d <db name> -i query.sql > query.advis.txt
```

This will generate a report about recommended or unused indexes.

Here's an example output:

Table 5: db2 advisor sample output

```
execution started at timestamp 2008-05-20-21.08.23.484000
found [1] SQL statements from the input file
Recommending indexes...
total disk space needed for initial set [   0.048] MB
total disk space constrained to         [  25.663] MB
Trying variations of the solution set.
Optimization finished.
  1  indexes in current solution
 [369.0000] timerons  (without recommendations)
```

[9] Note that the "explain tables" may need to be created. In order to do so, switch to the MISC directory of your DB2 installation, connect to the BPEDB and execute `db2 –tf EXPLAIN.DDL`.

```
 [ 13.0000] timerons  (with current solution)
 [96.48%] improvement



--
--
-- LIST OF RECOMMENDED INDEXES
-- ===========================
-- index[1],    0.048MB
   CREATE INDEX "EHTEST  "."IDX805201908260000" ON "EHTEST  "."TASK_INSTANCE_T"
("KIND" ASC, "STATE" ASC, "TKIID" ASC) ALLOW REVERSE SCANS ;
   COMMIT WORK ;
   RUNSTATS ON TABLE "EHTEST  "."TASK_INSTANCE_T" FOR INDEX "EHTEST
"."IDX805201908260000" ;
   COMMIT WORK ;



--
--
-- RECOMMENDED EXISTING INDEXES
-- ============================
-- RUNSTATS ON TABLE "EHTEST  "."TASK_INSTANCE_T" FOR INDEX "EHTEST  "."TI_STATE" ;
-- COMMIT WORK ;



--
--
-- UNUSED EXISTING INDEXES
-- ============================
-- DROP INDEX "EHTEST  "."TI_PARENT";
-- DROP INDEX "EHTEST  "."TI_ACOID";
-- DROP INDEX "EHTEST  "."TI_NAME";
-- DROP INDEX "EHTEST  "."TI_SERVICET";
-- DROP INDEX "EHTEST  "."TI_CCID";
-- DROP INDEX "EHTEST  "."TI_TK_TOPTK";
-- DROP INDEX "EHTEST  "."TI_TOPTKIID";
-- ===========================
--

7 solutions were evaluated by the advisor
DB2 Workload Performance Advisor tool is finished.
39 solutions were evaluated by the advisor
DB2 Workload Performance Advisor tool is finished.
```

Copy the text which is listed under `LIST OF RECOMMENDED INDEXES` into a new file `createIndexes.sql` and execute it, for instance:

```
db2 -f createIndexes.sql
```

Table 6: db2 advisor - section for index creation

```
-- LIST OF RECOMMENDED INDEXES
-- ===========================
```

```
-- index[1],    0.048MB
   CREATE INDEX "EHTEST  "."IDX805201908260000" ON "EHTEST  "."TASK_INSTANCE_T"
("KIND" ASC, "STATE" ASC, "TKIID" ASC) ALLOW REVERSE SCANS ;
   COMMIT WORK ;
   RUNSTATS ON TABLE "EHTEST  "."TASK_INSTANCE_T" FOR INDEX "EHTEST
"."IDX805201908260000" ;
   COMMIT WORK ;



--
--
-- RECOMMENDED EXISTING INDEXES
-- ============================
-- RUNSTATS ON TABLE "EHTEST  "."TASK_INSTANCE_T" FOR INDEX "EHTEST  "."TI_STATE" ;
-- COMMIT WORK ;
```

Note: The index recommendations from the DB2 advisor are typically improving query performance – we did not see cases where they did not help. However, the set of indexes may not provide the best performance which *can* be achieved. If you want to know the very best available set of indexes, request a DB2 consultant for manual SQL statement analysis and index recommendations. After the creation of indexes, update the statistics again as described earlier in this section.

5.5.2 Measurement Methodologies

In order to measure BPC query response times in a performance benchmarking scenario or a production scenario, the test driver must properly simulate the conditions during production. However, this is a difficult task and mostly not perfectly achievable due to its complexity.

An alternative approach for coarse grained measurements of BPC query response times in test environments is to turn on com.ibm.bpe.*=all tracing on WPS as described in chapter 5.6. It is important to know that if you turn on tracing, all other activities (except queries) will slow down significantly. Therefore, load tests with tracing do not represent the performance of your system – but queries do, with the constraint that you will only be able to see the BPC query performance.

Here is some sample trace output:

```
[4/4/08 11:45:03:109 ] >   com.ibm.bpe.database.Tom.setDataSource(Tom.java:245) ENTRY
...
[4/4/08 11:45:03:109 ] 3   com.ibm.bpe.database.Tom.queryWorkItem(Tom.java:8752)
DISTINCT TASK.TKIID, TASK_DESC.DISPLAY_NAME, …
[4/4/08 11:45:03:109 ] 3   com.ibm.bpe.database.Tom.queryWorkItem(Tom.java:8753)
(TASK.KIND IN(105,101,102) AND TASK.STATE IN( 2,8,1) AND NOT ( TASK.STATE IN (8) AND
WORK_ITEM.REASON IN (1)) …
[4/4/08 11:45:03:109 ] 3   com.ibm.bpe.database.Tom.queryWorkItem(Tom.java:8754)
TASK.TKIID
...
[4/4/08 11:45:03:359 ] 3
com.ibm.bpe.database.Tom.afterCompletionKeepCache(Tom.java:1236) Templates: 0
[4/4/08 11:45:03:359 ] <   com.ibm.bpe.database.Tom.afterCompletion(Tom.java:1215)
RETURN
```

Roughly, the time difference between the very first line of the sample above (…`Tom.setDataSource(…) ENTRY`) and the last line of the sample above (…`Tom.afterCompletion(…) RETURN`) is the time the query took (including any JDBC times which can be neglected most of the times[10]). For BPC queries that took more than 1 second before, this way of measurement gives a first indication whether the applied tuning was effective or not.

For more accurate BPC query response time measurements, you can use SQL statement snapshots on DB2 as described in chapter 5.5.2.1.

5.5.2.1 Using DB2 snapshots

DB2 snapshots provide insight to the resources needed by DB2 in order to execute a particular set of actions within a given time frame. A dynamic SQL snapshot for instance provides the following information:

- average query response times
- sort overflows
- bufferpool access statistics
- other

To some extent, a DB2 snapshot complements response time information which can also be measured with com.ibm.bpe tracing (but does not include the time spent on the JDBC driver and on the network). A database administrator can use the additional information in order to do more sophisticated performance tuning with respect to database memory utilization, sort heap etc.

In order to retrieve a dynamic SQL snapshot, connect to BPEDB and execute commands:

```
db2 update monitor switches using statement on
db2 update monitor switches using timestamp on
db2 reset monitor all
db2 FLUSH PACKAGE CACHE DYNAMIC
```

Then, execute your test case (focus on queries). In order to get a snapshot, execute:

```
db2 get snapshot for dynamic SQL on BPEDB > snapshot.txt
```

File `snapshot.txt` will look similar to the following sample output:

Table 7: Sample dynamic SQL snapshot on DB2

```
     Dynamic SQL Snapshot Result

Database name                       = BPEDB27

Database path                       = D:\DB2\NODE0000\SQL00003\
```

[10] In fact, a great latency between the WPS server and the database server can lead to performance problems. BPC is a database intensive application and therefore must be able to communicate over a fast network with the database.

```
...

 Number of executions              = 1
 Number of compilations            = 1
 Worst preparation time (ms)       = 1
 Best preparation time (ms)        = 1
 Internal rows deleted             = 0
 Internal rows inserted            = 0
 Rows read                         = 0
 Internal rows updated             = 0
 Rows written                      = 0
 Statement sorts                   = 0
 Statement sort overflows          = 0
 Total sort time                   = 0
 Buffer pool data logical reads    = Not Collected
 Buffer pool data physical reads   = Not Collected
 Buffer pool temporary data logical reads   = Not Collected
 Buffer pool temporary data physical reads  = Not Collected
 Buffer pool index logical reads   = Not Collected
 Buffer pool index physical reads  = Not Collected
 Buffer pool temporary index logical reads  = Not Collected
 Buffer pool temporary index physical reads = Not Collected
 Total execution time (sec.ms)     = 0.001251
 Total user cpu time (sec.ms)      = 0.000000
 Total system cpu time (sec.ms)    = 0.000000
 Statement text                    = select count(*) from task with ur
 ...
```

Note:

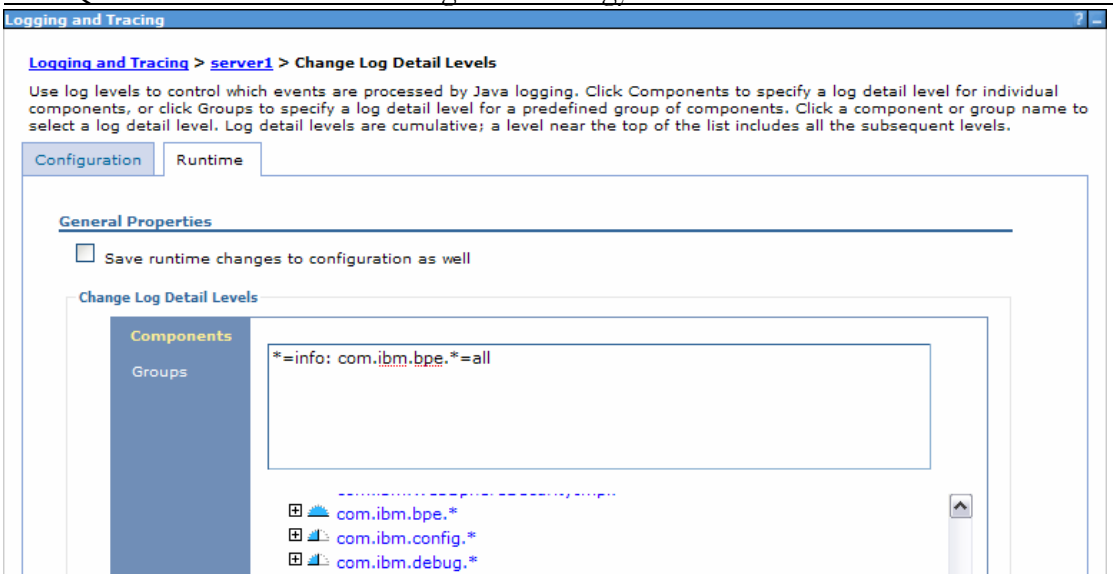
- The average response time of the query is "Total execution time (sec.ms)" divided by "Number of executions".
- In order to get locking and sorting statistics (not collected in the sample above), turn on the LOCK and SORT monitor in addition to the timestamp and statement monitor.
- The last line in Table 7 shows an SQL statement which has been executed.

5.6 G: Gather SQL Queries

BPC queries that require performance tuning should be tuned with database optimizers in order to get best response time. Before you can do that you need to find out what exact queries are issued against the system. To gather the queries, the best option is to use the mechanisms your DBMS is offering. On DB2, snapshots show the exact SQL statements that are executed by your BPC API queries (see chapter 5.5.2.1).

In a test environment you can also use the WPS built-in trace facility to gather the queries if you don't have immediate access to the underlying database system. Please note that turning on trace has a substantial effect on the overall system performance and thus is not applicable for production environment unless error situations must be handled.

To use the WPS trace to gather queries you have to first enable tracing. You do this in the WebSphere administrative console, as shown in the figure below, by turning on com.ibm.bpe.*=all. After you did that don't forget to press apply:



After the com.ibm.bpe.*=all trace is enabled, run the workload that executes the queries you are interested in, and which require to be tuned. For instance, run your GUI which shows the task lists for a user. Then, in the trace, look for `com.ibm.bpe.database.Query.<init>`:

```
com.ibm.bpe.database.Tom.queryWorkItem(Tom.java:8752) DISTINCT TASK.TKIID,
TASK_DESC.DISPLAY_NAME, TASK_DESC.DESCRIPTION, TASK_DESC.LOCALE, TASK.NAME
com.ibm.bpe.database.Tom.queryWorkItem(Tom.java:8753) (TASK.KIND IN(105,101,102) AND
TASK.STATE IN( 2,8,1) AND NOT ( TASK.STATE IN (8) AND WORK_ITEM.REASON IN (1)) AND
WORK_ITEM.REASON IN (4,1,5)) AND (TASK_DESC.LOCALE LIKE 'en%' OR TASK_DESC.LOCALE LIKE
'EN%' OR TASK_DESC.LOCALE = 'default')
com.ibm.bpe.database.Tom.queryWorkItem(Tom.java:8754) TASK.TKIID
com.ibm.bpe.database.Tom.queryWorkItem(Tom.java:8755) null
com.ibm.bpe.database.Tom.queryWorkItem(Tom.java:8756) null
com.ibm.bpe.database.Tom.queryWorkItem(Tom.java:8757) null
com.ibm.bpe.database.Tom.queryWorkItem(Tom.java:8758) userName = adjusters_0_0,
groupNames = [adjusters_0]
com.ibm.bpe.database.Tom.queryWorkItem(Tom.java:8759) true
…
com.ibm.bpe.database.Query.<init>(Query.java:73) ENTRY
com.ibm.bpe.database.Query.<init>(Query.java:210) SELECT DISTINCT TA.TKIID ,
TAD.DISPLAY_NAME , TAD.DESCRIPTION , TAD.LOCALE , TA.NAME  FROM TASK TA LEFT JOIN
TASK_DESC TAD ON (TA.TKIID = TAD.TKIID), WORK_ITEM WI WHERE (WI.OBJECT_ID = TA.TKIID)
AND (((TA.KIND IN (? ,? ,? )AND TA.STATE IN (? ,? ,? )AND NOT (TA.STATE IN (? )AND
WI.REASON IN (? ))AND WI.REASON IN (? ,? ,? ))AND (TAD.LOCALE LIKE ? OR TAD.LOCALE
LIKE ? OR TAD.LOCALE =? ))AND (WI.OWNER_ID =? OR (WI.OWNER_ID IS NULL AND
(WI.EVERYBODY =? OR WI.GROUP_NAME IN (? ))))) ORDER BY TA.TKIID  WITH UR
com.ibm.bpe.database.Query.<init>(Query.java:213) parm(0) = 105
com.ibm.bpe.database.Query.<init>(Query.java:213) parm(1) = 101
…
com.ibm.bpe.database.Query.<init>(Query.java:213) parm(14) = adjusters_0_0
com.ibm.bpe.database.Query.<init>(Query.java:213) parm(15) = 1
com.ibm.bpe.database.Query.<init>(Query.java:213) parm(16) = adjusters_0
com.ibm.bpe.database.Query.<init>(Query.java:215) RETURN
```

The bold marked text in the table above is the SQL string that is passed to the database, a few lines above you'll find the original strings (select clause, where clause, orderBy clause) passed to the BPC Query API.

Walk through the trace and look for the queries you want to tune. Copy the SQL queries to a separate document for later use.

Note: You will find parameter markers '? in the SQL queries, so those SQL queries are not directly executable. Nevertheless, the optimizer can use these queries in order to make recommendations for indexes or better access plans.

5.7 A2: Optimize query tables of implementation type DB Views

Database views which have been created in advanced tuning step (A1) are defined through SQL. SQL offers various optimization techniques. Make sure, that all possible optimization techniques have been used before another implementation type of query tables is being used. Consult your DB administrator or database expert in order to get advice for better SQL.

5.8 A3: Apply Materialized Views

Since WPS 6.0.2.1, materialized views are offered as an optimization technique for task and process list queries. Materialized views are described detailed in [MatViews].

Materialized views can be used perfectly together with tuning step (A1). If tuning step (A1) and (A2) did not provide good query performance, materialized views can help in case that you can afford your query to return slightly out-dated information (as configured with the *updateInterval* of the materialized view).

If you have a particular query that retrieves the task or process list, configure this query to be materialized as described in [MatViews]. This query can refer to a query table as explained in tuning step (A1). Note that this step only requires configuration, and no change in the application code or any change on the database views.

Note: If materialized views are used, it is important that the number of rows in the resulting materialized view are limited as good as possible:

- In your defining query, be as strict as possible.
- Staff resolution should resolve to a minimum number of work items for objects which will be contained in the materialized view (a group work item (verb "Group") results in a single work item only – so this is a best practice if you have group assignments).

5.9 A4: Apply query tables of implementation type DB table

In general, implementation type "DB tables" should not be used in order to speed up task or process list queries. The reason is, that a physical database table is hard to maintain without having in-depth knowledge about BPC processing and the BPC database schema.

However, in few situations, this might be the only choice left in order to get sufficient fast query response times. There is a variety of options to maintain a physical table which represents the contents of a BPC query:

- Using Java Snippets in BPEL processes. Custom properties (or Query Properties or Process Attributes) can be added to the DB table at process start or once the properties (as mentioned) are initialized.

- Using APIEventHandlers as described in [HTMAPI]
- Other

In case that the options given above are not sufficient or too complicated, consult the authors of this document in order to get further advice and to discuss other possible options.

Appendix A References

[Improving the performance of complex BPC API queries on DB2]

http://www-1.ibm.com/support/docview.wss?rs=2307&uid=swg21299450

[CustomTables]

http://www-1.ibm.com/support/docview.wss?uid=swg27010849

(for custom tables, see chapter 11)

[OptimizeQueries]

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.bpc.610.doc/doc/bpc/t5tuneproc_queries.html

[MatViews]

http://www-1.ibm.com/support/docview.wss?rs=2307&uid=swg27009623

[BFMAPI]

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/index.jsp?topic=/com.ibm.websphere.wbpmcore.javadoc.610.doc/web/apidocs/com/ibm/bpe/api/BusinessFlowManagerService.html

[HTMAPI]

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/index.jsp?topic=/com.ibm.websphere.wbpmcore.javadoc.610.doc/web/apidocs/com/ibm/task/api/HumanTaskManagerService.html

[BPCVIEWS]

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r1mx/topic/com.ibm.websphere.bpc.612.doc/doc/bpc/r6bpc_dbviews.html

[WPS62QueryTables]

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp?topic=/com.ibm.websphere.bpc.620.doc/doc/bpc/c6bpel_query_busproctask.html

Appendix B DB2 UDB statistics SQL

To gather DB2 statistics on BPC tables:

a) Connect to the BPEDB database:

```
db2 connect to BPEDB
```

b) Create the SQL script for executing runstats: in a DB2 command-line do:

```
db2 -x "select ' runstats on table ' concat rtrim(tabschema) concat
'.' concat tabname concat ' with distribution and detailed indexes
all ' from syscat.tables where type='T' AND tabname not in
('SAVED_ENGINE_MESSAGE_B_T') AND TBSPACEID IN (select TBSPACEID from
sysibm.systablespaces where TBSPACE IN ('INSTANCE', 'WORKITEM',
'STAFFQUERY','AUDITLOG', 'SCHEDTS')) " >> runStatsScript.sql
```

The select clause "`IN ('INSTANCE', ..., 'SCHEDTS')`" contains the default table spaces that are created and used when creating the BPEDB database. In case that in your environment the BPC tables are located in different table spaces, change this sub select accordingly.

c) Execute the generated SQL script (runStatsScript.sql):

```
db2 -f runStatsScript.sql
```

A sample output (the schema of the BPEDB has been replaced with @SCHEMA@) is listed in the table below:

```
runstats on table @SCHEMA@.AUDIT_LOG_T with distribution and detailed indexes all
runstats on table @SCHEMA@.TASK_AUDIT_LOG_T with distribution and detailed indexes all
runstats on table @SCHEMA@.PROCESS_INSTANCE_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.SCOPE_INSTANCE_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.ACTIVITY_INSTANCE_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.VARIABLE_INSTANCE_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.SCOPED_VARIABLE_INSTANCE_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.STAFF_MESSAGE_INSTANCE_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.EVENT_INSTANCE_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.REQUEST_INSTANCE_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.PARTNER_LINK_INSTANCE_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.VARIABLE_SNAPSHOT_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.CORR_SET_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.SUSPENDED_MESSAGE_INSTANCE_B_T with distribution and detailed indexes
all
runstats on table @SCHEMA@.CROSSING_LINK_INSTANCE_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.INVOKE_RESULT_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.INVOKE_RESULT2_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.EVENT_HANDLER_INSTANCE_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.CORRELATION_SET_INSTANCE_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.CORRELATION_SET_PROPERTIES_B_T with distribution and detailed indexes
all
runstats on table @SCHEMA@.UNDO_ACTION_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.CUSTOM_STMT_INSTANCE_B_T with distribution and detailed indexes all
```

```
runstats on table @SCHEMA@.COMP_WORK_PENDING_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.COMP_PARENT_ACTIVITY_INST_B_T with distribution and detailed indexes
all
runstats on table @SCHEMA@.RESTART_EVENT_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.PROCESS_INSTANCE_ATTRIBUTE_T with distribution and detailed indexes all
runstats on table @SCHEMA@.PROCESS_CONTEXT_T with distribution and detailed indexes all
runstats on table @SCHEMA@.ACTIVITY_INSTANCE_ATTR_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.NAVIGATION_EXCEPTION_T with distribution and detailed indexes all
runstats on table @SCHEMA@.AWAITED_INVOCATION_T with distribution and detailed indexes all
runstats on table @SCHEMA@.WORK_LIST_T with distribution and detailed indexes all
runstats on table @SCHEMA@.STORED_QUERY_T with distribution and detailed indexes all
runstats on table @SCHEMA@.FOR_EACH_INSTANCE_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.QUERYABLE_VARIABLE_INSTANCE_T with distribution and detailed indexes
all
runstats on table @SCHEMA@.ESCALATION_INSTANCE_T with distribution and detailed indexes all
runstats on table @SCHEMA@.ESC_INST_CPROP_T with distribution and detailed indexes all
runstats on table @SCHEMA@.ESC_INST_LDESC_T with distribution and detailed indexes all
runstats on table @SCHEMA@.EIDOCUMENTATION_T with distribution and detailed indexes all
runstats on table @SCHEMA@.ISERVICE_DESCRIPTION_T with distribution and detailed indexes all
runstats on table @SCHEMA@.TASK_INSTANCE_T with distribution and detailed indexes all
runstats on table @SCHEMA@.TASK_CONTEXT_T with distribution and detailed indexes all
runstats on table @SCHEMA@.CONTACT_QUERIES_T with distribution and detailed indexes all
runstats on table @SCHEMA@.UISETTINGS_T with distribution and detailed indexes all
runstats on table @SCHEMA@.REPLY_HANDLER_T with distribution and detailed indexes all
runstats on table @SCHEMA@.TASK_TIMER_T with distribution and detailed indexes all
runstats on table @SCHEMA@.TASK_INST_CPROP_T with distribution and detailed indexes all
runstats on table @SCHEMA@.TASK_INST_LDESC_T with distribution and detailed indexes all
runstats on table @SCHEMA@.TIDOCUMENTATION_T with distribution and detailed indexes all
runstats on table @SCHEMA@.ITASK_MESSAGE_DEFINITION_T with distribution and detailed indexes all
runstats on table @SCHEMA@.TASK_MESSAGE_INSTANCE_T with distribution and detailed indexes all
runstats on table @SCHEMA@.IMAIL_T with distribution and detailed indexes all
runstats on table @SCHEMA@.MV_CTR_T with distribution and detailed indexes all
runstats on table @SCHEMA@.NAVIGATION_CLEANUP_TIMER_B_T with distribution and detailed indexes all
runstats on table @SCHEMA@.WORK_ITEM_T with distribution and detailed indexes all
runstats on table @SCHEMA@.WI_ASSOC_OID_T with distribution and detailed indexes all
runstats on table @SCHEMA@.RETRIEVED_USER_T with distribution and detailed indexes all
runstats on table @SCHEMA@.SCHED_TASK with distribution and detailed indexes all
runstats on table @SCHEMA@.SCHED_TREG with distribution and detailed indexes all
runstats on table @SCHEMA@.SCHED_LMGR with distribution and detailed indexes all
runstats on table @SCHEMA@.SCHED_LMPR with distribution and detailed indexes all
```

Appendix C Artifacts used by the motivating example

The following sections provide details on which artifacts are used for the different steps T0 – T3 of the motivating example in section 2.

C.1 T0 (No tuning)

Table 8: Query java code snippet

```
// htm is the HumanTaskManager EJB interface
htm.query(

// select clause
"TASK.TKIID, QUERY_PROPERTY0.STRING_VALUE, QUERY_PROPERTY1.STRING_VALUE, …,
QUERY_PROPERTY9.STRING_VALUE",


// where clause
"TASK.STATE=TASK.STATE.STATE_READY AND QUERY_PROPERTY0.NAME='property0' AND
QUERY_PROPERTY1.NAME='property1' AND … AND QUERY_PROPERTY9.NAME='property9'",

// order by clause
null,
…
```

Table 9: Resulting SQL

```
SELECT
        TA.TKIID,
        QP0.STRING_VALUE,
        QP1.STRING_VALUE,
        …
        QP9.STRING_VALUE
FROM
        TASK TA,
        QUERY_PROPERTY QP0,
        QUERY_PROPERTY QP1,
        …
        QUERY_PROPERTY QP9,
        WORK_ITEM WI
WHERE
        TA.STATE=2 AND
        QP0.NAME='property0' AND
        QP1.NAME='property1' AND
        …
        QP9.NAME='property9' AND
        (WI.OWNER_ID='<owner>' OR WI.OWNER_ID IS null AND (WI.GROUP_NAME IN
('<groups>') OR WI.EVERY BODY=1))
```

C.2 T1 (Best practice)

Table 10: Query java code snippet

```
// htm is the HumanTaskManager EJB interface
htm.query(

// select clause
"MY_TASKS.TKIID, MY_TASKS.PROP0, MY_TASKS.PROP1, …, MY_TASKS.PROP9",


// where clause (defined in the database view)
null,

// order by clause
null,
…
```

Table 11: Custom view XML (for registration within BPC)

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<customtable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://www.ibm.com/schemas/workflow/wswf/customtable"
        xsi:schemaLocation="http://www.ibm.com/schemas/workflow/wswf/customtable
http://www.ibm.com/schemas/workflow/wswf/customtable">
        <querytableinfo tablename="MCT" aliasname="MCT" joinlevel="3">
                <joincolumn column="TA_TKIID" />
                <joincolumn column="TA_TKIID" target="WORK_ITEM" />
                <querycolumninfo columnname="TKIID" type="TYPE_ID" isNullable="false"
/>
        <querycolumninfo columnname="PROP0" type="TYPE_STRING" isNullable="true" />
        <querycolumninfo columnname="PROP1" type="TYPE_STRING" isNullable="true" />
        <querycolumninfo columnname="PROP2" type="TYPE_STRING" isNullable="true" />
        <querycolumninfo columnname="PROP3" type="TYPE_STRING" isNullable="true" />
        <querycolumninfo columnname="PROP4" type="TYPE_STRING" isNullable="true" />
        <querycolumninfo columnname="PROP5" type="TYPE_STRING" isNullable="true" />
        <querycolumninfo columnname="PROP6" type="TYPE_STRING" isNullable="true" />
        <querycolumninfo columnname="PROP7" type="TYPE_STRING" isNullable="true" />
        <querycolumninfo columnname="PROP8" type="TYPE_STRING" isNullable="true" />
        <querycolumninfo columnname="PROP9" type="TYPE_STRING" isNullable="true" />
        </querytableinfo>
</customtable>
```

Table 12: Custom view definition (DDL for creation in the BPEDB)

```
CREATE VIEW MY_TASK
(TKIID, PROP0, PROP1, …, PROP9) AS
SELECT
        TA.TKIID,
        QP0.STRING_VALUE,
        QP1.STRING_VALUE,
        …
        QP9.STRING_VALUE
FROM
        TASK TA,
        QUERY_PROPERTY QP0,
        QUERY_PROPERTY QP1,
        …
        QUERY_PROPERTY QP9
WHERE
        TA.STATE=2 AND
        QP0.NAME='property0' AND
        QP1.NAME='property1' AND
        …
        QP9.NAME='property9'
```

Table 13: Resulting SQL

```
SELECT
        MY_TASKS.TKIID,
        MY_TASKS.PROP0,
        MY_TASKS.PROP1,
        …
        MY_TASKS.PROP9
FROM
        MY_TASKS MY_TASK,
        WORK_ITEM WI
WHERE
        (WI.OWNER_ID='<owner>' OR WI.OWNER_ID=null AND (WI.GROUP_NAME IN ('<groups>')
OR WI.EVERY_BODY=1))
```

C.3 T2 (Advanced tuning)

Differences to previous test cases shown only:

Table 14: Tuned custom view definition (DDL for creation in the BPEDB)

```
CREATE VIEW MY_TASK
(TKIID, PROP0, PROP1, …, PROP9) AS
SELECT
        TA.TKIID,
        (SELECT QP.STRING_VALUE FROM QUERY_PROPERTY QP
         WHERE QP.PIID=TA.CONTAINMENT_CTX_ID AND QP.NAME='property0'),
        (SELECT QP.STRING_VALUE FROM QUERY_PROPERTY QP
         WHERE QP.PIID=TA.CONTAINMENT_CTX_ID AND QP.NAME='property1'),
        …
```

```
        (SELECT QP.STRING_VALUE FROM QUERY_PROPERTY QP
         WHERE QP.PIID=TA.CONTAINMENT_CTX_ID AND QP.NAME='property9'),
FROM
        TASK TA
WHERE
        TA.STATE=2
```

C.4 T3 (No authorization)

Differences to previous test cases shown only:

Table 15: Query java code snippet

```
// htm is the HumanTaskManager EJB interface
htm.queryAll(

// select clause
"MY_TASKS.TKIID, MY_TASKS.PROP0, MY_TASKS.PROP1, …, MY_TASKS.PROP9",


// where clause (defined in the database view)
null,

// order by clause
null,
…
```

Appendix D Use of scalar fullselects with the BPC schema

Scalar fullselects is a SQL technique which is available on DB2 and Oracle. For a detailed description, see the related product documentation.

A **scalar fullselect** is a select statement that is used in the select-clause of a SQL query. For the scalar fullselect, there must only 1 row qualify for each row that is returned by the SQL query. Example for DB2:

```
SELECT
      TKIID,
      STATE,
      (SELECT
            NAME
            FROM PROCESS_INSTANCE PI
            WHERE TA.CONTAINMENT_CTX_ID = PI.PIID
      )
      FROM TASK TA WITH UR
```

SQL accessing data in the BPC schema is best written with scalar fullselects, considering the following hint:

- Views that provide additional information to the selected objects (but do not occur in the where-clause which restricts the total number of rows to be returned), should be accessed using scalar fullselects.

Typically, a BPC query which needs performance tuning queries a list of objects – mainly human tasks (view TASK) or process instances (view PROCESS_INSTANCE). The relationship between these *primary views* to other views (the *attached views*) is a one-to-(zero or one) relationship. This means: a specific task or process instance typically only occurs once in the result set; additional data (such as query properties or custom properties) may be available for a specific task or not. Appendix E lists the most common combinations in BPC queries, between the TASK view (as *primary view*) and the PROCESS_INSTANCE view (as *primary view*) and other views (as *attached views*). The table contains the join conditions which must be in place in order to express the one-to-(zero or one) relationship. For instance, if the TASK view is joined with the QUERY_PROPERTY view, the name of the QUERY_PROPERTY must be contained in the join condition in order to establish the one-to-(zero or one) relationship.

Appendix E Join Columns

The description of the examples in this section follows the following pattern:

	Explanation	
Join of *VIEW* (*ALIAS*) with…	**… Attached View**	**Rule #**
VIEW refers to the name of the primary view, *ALIAS* is the alias used in join conditions displayed in this column. This join condition must be used in order to preserve the one-to-(zero or one) relationship between the primary view and the attached view.	The name of the attached view, along with its alias in brackets.	The rule which will be referred to later in this document (Appendix F)
…	…	…

Example:

Join of PROCESS_INSTANCE (PI) with…	**… Attached View**	**Rule #**
`(PI.PIID=AI.PIID) AND (AI.NAME=?)`	`ACTIVITY (AI)`	P1

If PROCESS_INSTANCE is chosen as primary view, and ACTIVITY is attached to it, the following SQL using scalar fullselects would be valid in the context of this document:

```
SELECT
      PI.NAME,
      PI.PIID,

      -- scalar fullselect start
      (
            SELECT AI.STATE FROM ACTIVITY AI
            -- preserve one-to-(zero or one) relationship
            WHERE (PI.PIID=AI.PIID) AND (AI.NAME='test')
      )
      -- scalar fullselect end

FROM
      PROCESS_INSTANCE PI
```

E.1 Primary View: Process Instance

Join of PROCESS_INSTANCE (PI) with…	… Attached View	Rule #
(PI.PIID=AI.PIID) AND (AI.NAME=?)	ACTIVITY (AI)	P1
(PI.PIID=AI.PIID) AND (AI.AIID=AIA.AIID) AND (AIA.NAME=?)	ACTIVITY_ ATTRIBUTE (AIA)	P2
(PI.PIID=PAT.PIID) AND (PAT.NAME=?)	PROCESS_ ATTRIBUTE (PAT)	P3
(PI.PTID=PT.PTID)	PROCESS_ TEMPLATE (PT)	P4
(PI.PIID=QP.PIID) AND (QP.NAME=?)	QUERY_ PROPERTY (QP)	P5
(PI.PIID=TA.CONTAINMENT_CTX_ID) AND (TA.NAME=?)	TASK (TA)	P6
(PI.PIID=TA.CONTAINMENT_CTX_ID) AND (TA.NAME=?) AND (TA.TKIID=TCP.TKIID) AND (TCP.NAME=?)	TASK_ CPROP (TCP)	P7
(PI.PIID=TA.CONTAINMENT_CTX_ID) AND (TA.TKIID=TD.TKIID) AND (TA.NAME=?)	TASK_ DESC (TD)	P8
PI.PIID=TA.CONTAINMENT_CTX_ID) AND (TA.TKTID=TT.TKTID) AND (TA.NAME=?)	TASK_ TEMPL (TT)	P9
(PI.PIID=TA.CONTAINMENT_CTX_ID) AND (TA.TKTID=TTC.TKTID) AND (TA.NAME=?)	TASK_ TEMPL_CPROP (TTC)	P10
(PI.PIID=TA.CONTAINMENT_CTX_ID) AND (TA.TKTID=TTD.TKTID) AND (TA.NAME=?) AND (TTD.LOCALE=?)	TASK_ TEMPL_DESC (TTD)	P11

E.2 Primary View: Task

Join of TASK (TA) with…	… Attached View	Rule
(TA.PARENT_CONTEXT_ID=AI.AIID)	ACTIVITY (AI)	P1
(TA.PARENT_CONTEXT_ID=AI.AIID) AND (AIA.NAME=?)	ACTIVITY_ ATTRIBUTE (AIA)	P2
(TA.CONTAINMENT_CTX_ID=PAT.PIID) AND (PAT.NAME=?)	PROCESS_ ATTRIBUTE (PAT)	P3
(TA.CONTAINMENT_CTX_ID=PI.PIID)	PROCESS_ INSTANCE (PI)	P4
(TA.CONTAINMENT_CTX_ID=PI.PIID) AND (PI.PTID=PT.PTID)	PROCESS_ TEMPLATE (PT)	P5
(TA.CONTAINMENT_CTX_ID=QP.PIID) AND (QP.NAME=?)	QUERY_ PROPERTY (QP)	P6
(TA.TKIID=TCP.TKIID) AND (TCP.NAME=?)	TASK_ CPROP (TCP)	P7
(TA.TKIID=TD.TKIID) AND (TD.LOCALE=?)	TASK_ DESC (TD)	P8
(TA.TKTID=TT.TKTID)	TASK_ TEMPL (TT)	P9
(TA.TKTID=TTC.TKTID) AND (TTC.NAME=?)	TASK_ TEMPL_CPROP (TTC)	P10
(TA.TKTID=TTD.TKTID) AND (TTD.LOCALE=?)	TASK_ TEMPL_DESC (TTD)	P11

Appendix F Examples

The description of the examples in this section follows the following pattern:

	Explanation
Description	Describes the scenario, a short introduction.
Primary View	The primary view in this scenario (this must be exactly one).
Attached Views	The attached views in this scenario (may be many or none).
Referenced BPC Views	The list of all views involved – primary views and attached views. Note that the WORK_ITEM view is not shown here, because its usage is dependent on the interface used for the query (query(…) vs. queryAll(…)).
Rules	The rules which have been used to create the join conditions, as defined in Appendix E.
HTM Query	A sample query which queries the query tables defined in this scenario, using the built-in views.
SFS View	The views that are attached using scalar fullselects. Typically, all attached views can be found here. Under circumstances (e.g. if the task list is filtering by particular attached view information), attached views should not be added with scalar fullselects (but with standard joins then).
DB View	The view definition of the query table which uses a database view (query table of implementation type database view).
HTM Query against DB View as custom table	A sample query which queries the query table.
Custom Table Definition	The XML definition of the "custom table XML" definition, as needed to register the query table (custom table) with BPC.

F.1 Task lists

Task list examples in this chapter contain tasks which are filtered by properties on tasks only (like the task state). We talk about query tables with *primary view* TASK.

F.1.1 Example with BPC view TASK

Description	A simple task list which shows tasks in state ready.
Primary View	`TASK`
Attached Views	-
Referenced BPC Views	`TASK`
Rules	-
HTM Query	`htm.query("TASK.TKIID", "TASK.STATE=TASK.STATE.STATE_READY", …)`
SFS View	-
DB View	`CREATE VIEW MY_TASK_LIST (TKIID) AS SELECT TA.TKIID FROM TASK TA WHERE TA.STATE=2`
HTM Query against DB View as custom table	`htm.query("MY_TASK_LIST.TKIID")`
Custom Table Definition	`<?xml version="1.0" encoding="UTF-8"?> <customtable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.ibm.com/schemas/workflow/wswf/customtable" xsi:schemaLocation="http://www.ibm.com/schemas/workflow/wswf/customtable http://www.ibm.com/schemas/workflow/wswf/customtable"> <querytableinfo tablename="MY_TASK_LIST" aliasname="MTL" joinlevel="3"> <joincolumn column="TKIID" /> <joincolumn column="TKIID" target="WORK_ITEM" /> <querycolumninfo columnname="TKIID" type="TYPE_ID" isNullable="false" /> </querytableinfo> </customtable>`

F.1.2 Example with BPC views TASK, TASK_CPROP

Description	A simple task list which shows tasks in state ready, along with the task property "customer".
Primary View	`TASK`
Attached Views	`TASK_CPROP`
Referenced BPC Views	`TASK, TASK_CPROP`
Rules	`P7`
HTM Query	`htm.query( "TASK.TKIID, TASK_CPROP.STRING_VALUE", "TASK.STATE=TASK.STATE.STATE_READY AND TASK_CPROP.NAME='customer'", …)`
SFS View	`TASK_CPROP`
DB View	`CREATE VIEW MY_TASK_LIST (TKIID, CUSTOMER) AS SELECT TA.TKIID, (SELECT TCP.STRING_VALUE FROM TASK_CPROP TCP WHERE TCP.TKIID=TA.TKIID AND TCP.NAME='customer' ) FROM TASK TA WHERE TA.STATE=2`
HTM Query against DB View as custom table	`htm.query("MY_TASK_LIST.TKIID, MY_TASK_LIST.CUSTOMER")`
Custom Table Definition	`<?xml version="1.0" encoding="UTF-8"?> <customtable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.ibm.com/schemas/workflow/wswf/customtable" xsi:schemaLocation="http://www.ibm.com/schemas/workflow/wswf/customtable http://www.ibm.com/schemas/workflow/wswf/customtable"> <querytableinfo tablename="MY_TASK_LIST" aliasname="MTL" joinlevel="3"> <joincolumn column="TKIID" /> <joincolumn column="TKIID" target="WORK_ITEM" /> <querycolumninfo columnname="TKIID" type="TYPE_ID" isNullable="false" /> <querycolumninfo columnname="CUSTOMER" type="TYPE_STRING" isNullable="true" /> </querytableinfo> </customtable>`

F.1.3 Example with BPC views TASK, TASK_CPROP (two times)

Description	A simple task list which shows tasks in state ready, with two task properties: "customer" and "address".
Primary View	TASK
Attached Views	TASK_CPROP (2 times)
Referenced BPC Views	TASK, TASK_CPROP (2 times)
Rules	P7
HTM Query	htm.query("TASK.TKIID, TASK_CPROP1.STRING_VALUE, TASK_CPROP2.STRING_VALUE", "TASK.STATE=TASK.STATE.STATE_READY AND TASK_CPROP1.NAME='customer' AND TASK_CPROP2.NAME='address'", …)
SFS View	TASK_CPROP1, TASK_CPROP2
DB View	CREATE VIEW MY_TASK_LIST (TKIID, CUSTOMER, ADDRESS) AS SELECT TA.TKIID, (SELECT TCP.STRING_VALUE FROM TASK_CPROP TCP WHERE TCP.TKIID=TA.TKIID AND TCP.NAME='customer'), (SELECT TCP.STRING_VALUE FROM TASK_CPROP TCP WHERE TCP.TKIID=TA.TKIID AND TCP.NAME='address') FROM TASK TA WHERE TA.STATE=2
HTM Query against DB View as custom table	htm.query("MY_TASK_LIS.TKIID, MY_TASK_LIST.CUSTOMER, MY_TASK_LIST.ADDRESS")
Custom Table Definition	<?xml version="1.0" encoding="UTF-8"?> <customtable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.ibm.com/schemas/workflow/wswf/customtable" xsi:schemaLocation="http://www.ibm.com/schemas/workflow/wswf/customtable http://www.ibm.com/schemas/workflow/wswf/customtable"> <querytableinfo tablename="MY_TASK_LIST" aliasname="MTL" joinlevel="3"> <joincolumn column="TKIID" /> <joincolumn column="TKIID" target="WORK_ITEM" /> <querycolumninfo columnname="TKIID" type="TYPE_ID" isNullable="false" /> <querycolumninfo columnname="CUSTOMER" type="TYPE_STRING" isNullable="true" /> <querycolumninfo columnname="ADDRESS" type="TYPE_STRING" isNullable="true" /> </querytableinfo> </customtable>

F.1.4 Example with BPC views TASK, TASK_CPROP, QUERY_PROPERTY

Description	A simple task list which shows tasks in state ready, with one task property ("customer") and one query property ("address").
Primary View	TASK
Attached Views	TASK_CPROP, QUERY_PROPERTY
Referenced BPC Views	TASK, TASK_CPROP, QUERY_PROPERTY
Rules	P6, P7
HTM Query	htm.query("TASK.TKIID, TASK_CPROP.STRING_VALUE, QUERY_PROPERTY.STRING_VALUE", "TASK.STATE=TASK.STATE.STATE_READY AND TASK_CPROP.NAME='customer' AND QUERY_PROPERTY.NAME='address'", …)
SFS View	TASK_CPROP, QUERY_PROPERTY
DB View	CREATE VIEW MY_TASK_LIST (TKIID, CUSTOMER, ADDRESS) AS SELECT TA.TKIID, (SELECT TCP.STRING_VALUE FROM TASK_CPROP TCP WHERE TCP.TKIID=TA.TKIID AND TCP.NAME='customer'), (SELECT QP.STRING_VALUE FROM QUERY_PROPERTY QP WHERE QP.PIID=TA.CONTAINMENT_CTX_ID AND QP.NAME='address') FROM TASK TA WHERE TA.STATE=2
HTM Query against DB View as custom table	htm.query("MY_TASK_LIST.TKIID, MY_TASK_LIST.CUSTOMER, MY_TASK_LIST.ADDRESS")
Custom Table Definition	<?xml version="1.0" encoding="UTF-8"?> <customtable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.ibm.com/schemas/workflow/wswf/customtable" xsi:schemaLocation="http://www.ibm.com/schemas/workflow/wswf/customtable http://www.ibm.com/schemas/workflow/wswf/customtable"> <querytableinfo tablename="MTL" aliasname="MTL" joinlevel="3"> <joincolumn column="TKIID" /> <joincolumn column="TKIID" target="WORK_ITEM" /> <querycolumninfo columnname="TKIID" type="TYPE_ID" isNullable="false" /> <querycolumninfo columnname="CUSTOMER" type="TYPE_STRING" isNullable="true" /> <querycolumninfo columnname="ADDRESS" type="TYPE_STRING" isNullable="true" /> </querytableinfo> </customtable>

F.1.5 Example with BPC views TASK, PROCESS_INSTANCE

Description	A simple task list which shows tasks in state ready, and the process instance it belongs to
Primary View	TASK
Attached Views	PROCESS_INSTANCE
Referenced BPC Views	TASK, PROCESS_INSTANCE
Rules	P4
HTM Query	htm.query("TASK.TKIID, PROCESS_INSTANCE.NAME", "TASK.STATE=TASK.STATE.STATE_READY", …)
SFS View	PROCESS_INSTANCE
DB View	CREATE VIEW MY_TASK_LIST (TKIID, PI_NAME) AS SELECT TA.TKIID, (SELECT PI.NAME FROM PROCESS_INSTANCE PI WHERE PI.PIID=TA.CONTAINMENT_CTX_ID) FROM TASK TA WHERE TA.STATE=2
HTM Query against DB View as custom table	htm.query("MY_TASK_LIST.TKIID, MY_TASK_LIST.PI_NAME")
Custom Table Definition	<?xml version="1.0" encoding="UTF-8"?> <customtable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.ibm.com/schemas/workflow/wswf/customtable" xsi:schemaLocation="http://www.ibm.com/schemas/workflow/wswf/customtable http://www.ibm.com/schemas/workflow/wswf/customtable"> <querytableinfo tablename="MY_TASK_LIST" aliasname="MTL" joinlevel="3"> <joincolumn column="TKIID" /> <joincolumn column="TKIID" target="WORK_ITEM" /> <querycolumninfo columnname="TKIID" type="TYPE_ID" isNullable="false" /> <querycolumninfo columnname="PI_NAME" type="TYPE_STRING" isNullable="true" /> </querytableinfo> </customtable>

F.1.6 Example with BPC views TASK, TASK_DESC

Description	A simple task list which shows tasks in state ready, with its display name (default locale).
Primary View	`TASK`
Attached Views	`TASK_DESC`
Referenced BPC Views	`TASK, TASK_DESC`
Rules	`P8`

HTM Query	`htm.query(`
	`"TASK.TKIID,`
	`TASK_DESC.DISPLAY_NAME",`
	`"TASK.STATE=TASK.STATE.STATE_READY AND`
	`TASK_DESC.LOCALE='default'", …)`

SFS View	`TASK_DESC`

DB View	`CREATE VIEW MY_TASK_LIST`
	`(TKIID, DISPLAY_NAME)`
	`AS`
	`SELECT`
	`  TA.TKIID,`
	`  (SELECT TD.DISPLAY_NAME`
	`  FROM TASK_DESC TD`
	`  WHERE`
	`  TD.TKIID=TA.TKIID AND`
	`  TD.LOCALE='default'`
	`  )`
	`FROM TASK TA`
	`WHERE TA.STATE=2`

HTM Query against DB View as custom table	`htm.query("MY_TASK_LIST.TKIID,` `MY_TASK_LIST.DISPLAY_NAME")`

Custom Table Definition	`<?xml version="1.0" encoding="UTF-8"?>`
	`<customtable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`  xmlns="http://www.ibm.com/schemas/workflow/wswf/customtable"`
	`xsi:schemaLocation="http://www.ibm.com/schemas/workflow/wswf/customtable`
	`http://www.ibm.com/schemas/workflow/wswf/customtable">`
	`  <querytableinfo tablename="MY_TASK_LIST" aliasname="MTL"`
	`joinlevel="3">`
	`  <joincolumn column="TKIID" />`
	`  <joincolumn column="TKIID" target="WORK_ITEM" />`
	`  <querycolumninfo columnname="TKIID" type="TYPE_ID"`
	`isNullable="false" />`
	`  <querycolumninfo columnname="DISPLAY_NAME" type="TYPE_STRING"`
	`isNullable="true" />`
	`  </querytableinfo>`
	`</customtable>`

F.2 Process lists

Process list examples in this chapter contain process instances which are filtered by properties on process instances only (like the process instance state). We talk about query tables with *primary view* PROCESS_INSTANCE.

F.2.1 Example with BPC view PROCESS_INSTANCE

Description	A simple process list which shows all process instances.
Primary View	`PROCESS_INSTANCE`
Attached Views	-
Referenced BPC Views	`PROCESS_INSTANCE`
Rules	-
BFM Query	`bfm.query("PROCESS_INSTANCE.PIID", null, …)`
SFS View	-

DB View	`CREATE VIEW MY_PROCESS_LIST`
	`(PIID)`
	`AS`
	`SELECT PI.PIID`
	`FROM PROCESS_INSTANCE PI`

BFM Query against DB View as custom table	`bfm.query("MY_PROCESS_LIST.PIID")`

Custom Table Definition	`<?xml version="1.0" encoding="UTF-8"?>`
	`<customtable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`  xmlns="http://www.ibm.com/schemas/workflow/wswf/customtable"`
	`xsi:schemaLocation="http://www.ibm.com/schemas/workflow/wswf/customtable`
	`http://www.ibm.com/schemas/workflow/wswf/customtable">`
	`  <querytableinfo tablename="MY_PROCESS_LIST" aliasname="MPL"`
	`joinlevel="3">`
	`  <joincolumn column="PIID" />`
	`  <joincolumn column="PIID" target="WORK_ITEM" />`
	`  <querycolumninfo columnname="PIID" type="TYPE_ID"`
	`  isNullable="false" />`
	`  </querytableinfo>`
	`</customtable>`

F.2.2 Example with BPC views PROCESS_INSTANCE, ACTIVITY, ACTIVITY_ATTRIBUTE

Description	A simple process list which shows all process instances and an activity with a particular name.
Primary View	`PROCESS_INSTANCE`
Attached Views	`ACTIVITY, ACTIVITY_ATTRIBUTE`
Referenced BPC Views	`PROCESS_INSTANCE, ACTIVITY, ACTIVITY_ATTRIBUTE`
Rules	`P1`
BFM Query	`bfm.query( "PROCESS_INSTANCE.PIID, ACTIVITY.STATE", null, …)`
SFS View	-
DB View	`CREATE VIEW MY_PROCESS_LIST (PIID, A_STATE) AS SELECT PI.PIID, (SELECT AI.STATE FROM ACTIVITY AI WHERE AI.NAME='IdTracker' AND AI.PIID=PI.PIID) FROM PROCESS_INSTANCE PI`
BFM Query against DB View as custom table	`bfm.query("MY_PROCESS_LIST.PIID, MY_PROCESS_LIST.A_STATE")`
Custom Table Definition	`<?xml version="1.0" encoding="UTF-8"?> <customtable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.ibm.com/schemas/workflow/wswf/customtable" xsi:schemaLocation="http://www.ibm.com/schemas/workflow/wswf/customtable http://www.ibm.com/schemas/workflow/wswf/customtable"> <querytableinfo tablename="MY_PROCESS_LIST" aliasname="MPL" joinlevel="3"> <joincolumn column="PIID" /> <joincolumn column="PIID" target="WORK_ITEM" /> <querycolumninfo columnname="PIID" type="TYPE_ID" isNullable="false" /> <querycolumninfo columnname="A_STATE" type="TYPE_NUMBER" isNullable="false" /> </querytableinfo> </customtable>`

F.2.3 Example with BPC views PROCESS_INSTANCE, PROCESS_ATTRIBUTE, QUERY_PROPERTY

Description	A simple process list which shows all process instances and a process attribute and a query property.
Primary View	`PROCESS_INSTANCE`
Attached Views	`PROCESS_ATTRIBUTE, QUERY_PROPERTY`
Referenced BPC Views	`PROCESS_INSTANCE, PROCESS_ATTRIBUTE, QUERY_PROPERTY`
Rules	`P3, P5`

BFM Query	`bfm.query(`
	`"PROCESS_INSTANCE.PIID,`
	`PROCESS_ATTRIBUTE.VALUE,`
	`QUERY_PROPERTY.STRING_VALUE",`
	`"PROCESS_ATTRIBUTE.NAME='ID' AND`
	`QUERY_PROPERTY.NAME='Customer'", …)`

SFS View	-

DB View	`CREATE VIEW MY_PROCESS_LIST`
	`(PIID, ID, CUSTOMER)`
	`AS`
	`SELECT PI.PIID,`
	`(SELECT PA.VALUE FROM PROCESS_ATTRIBUTE PA`
	`WHERE PA.NAME='ID' AND PA.PIID=PI.PIID),`
	`(SELECT QP.STRING_VALUE FROM QUERY_PROPERTY QP`
	`WHERE QP.NAME='Customer' AND QP.PIID=PI.PIID)`
	`FROM PROCESS_INSTANCE PI`

BFM Query against DB View as custom table	`bfm.query("MY_PROCESS_LIST.PIID, MY_PROCESS_LIST.ID,` `MY_PROCESS_LIST.CUSTOMER")`

Custom Table Definition	`<?xml version="1.0" encoding="UTF-8"?>`
	`<customtable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`xmlns="http://www.ibm.com/schemas/workflow/wswf/customtable"`
	`xsi:schemaLocation="http://www.ibm.com/schemas/workflow/wswf/customtable`
	`http://www.ibm.com/schemas/workflow/wswf/customtable">`
	`<querytableinfo tablename="MY_PROCESS_LIST" aliasname="MPL"`
	`joinlevel="3">`
	`<joincolumn column="PIID" />`
	`<joincolumn column="PIID" target="WORK_ITEM" />`
	`<querycolumninfo columnname="PIID" type="TYPE_ID" isNullable="false"`
	`/>`
	`<querycolumninfo columnname="ID" type="TYPE_STRING"`
	`isNullable="false" />`
	`<querycolumninfo columnname="CUSTOMER" type="TYPE_STRING"`
	`isNullable="false" />`
	`</querytableinfo>`
	`</customtable>`

F.3 Advanced: Task and Process lists with specific criteria

List examples in this chapter contain tasks or process instances which are filtered not only by properties on process instances or tasks only (like state), but also by the existence of a particular property.

F.3.1 Example with BPC views TASK, TASK_CPROP (2 times)

Description	A simple task list which shows tasks in state ready, along with the task property "customer". The task list only contains human tasks which have a task property "color" of value "red".
Primary View	TASK
Attached View	TASK_CPROP (2 times)
Referenced BPC Views	TASK, TASK_CPROP
Rules	P7
HTM Query	htm.query("TASK.TKIID, TASK_CPROP.STRING_VALUE", "TASK.STATE=TASK.STATE.STATE_READY AND TASK_CPROP1.NAME='color' AND TASK_CPROP1.STRING_VALUE='red' AND TASK_CPROP2.NAME='customer'", …)
SFS View	TASK_CPROP
DB View	CREATE VIEW MY_TASK_LIST (TKIID, CUSTOMER) AS SELECT TA.TKIID, (SELECT TCP1.STRING_VALUE FROM TASK_CPROP TCP1 WHERE TCP1.TKIID=TA.TKIID AND TCP1.NAME='customer') FROM TASK TA, TASK_CPROP TCP2 WHERE TA.STATE=2 AND TCP2.NAME='color' AND TCP2.STRING_VALUE='red' AND TCP2.TKIID=TA.TKIID
HTM Query against DB View as custom table	htm.query("MY_TASK_LIST.TKIID, MY_TASK_LIST.CUSTOMER")
Custom Table Definition	<?xml version="1.0" encoding="UTF-8"?> <customtable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.ibm.com/schemas/workflow/wswf/customtable" xsi:schemaLocation="http://www.ibm.com/schemas/workflow/wswf/customtable http://www.ibm.com/schemas/workflow/wswf/customtable"> <querytableinfo tablename="MY_TASK_LIST" aliasname="MTL" joinlevel="3"> <joincolumn column="TKIID" /> <joincolumn column="TKIID" target="WORK_ITEM" /> <querycolumninfo columnname="TKIID" type="TYPE_ID" isNullable="false" /> <querycolumninfo columnname="CUSTOMER" type="TYPE_STRING" isNullable="true" /> </querytableinfo> </customtable>

F.3.2 Example with BPC views PROCESS_INSTANCE, PROCESS_ATTRIBUTE, QUERY_PROPERTY

Description	A simple process list which shows all process instances and a process attribute and a query property. The query property must have a specific value for the process instance being listed in the result.
Primary View	PROCESS_INSTANCE
Attached View	PROCESS_ATTRIBUTE, QUERY_PROPERTY
Referenced BPC Views	PROCESS_INSTANCE, PROCESS_ATTRIBUTE, QUERY_PROPERTY
Rules	P3, P5
BFM Query	bfm.query("PROCESS_INSTANCE.PIID, PROCESS_ATTRIBUTE.VALUE, QUERY_PROPERTY.STRING_VALUE", "PROCESS_ATTRIBUTE.NAME='ID' AND QUERY_PROPERTY.NAME='Customer' AND QUERY_PROPERTY.STRING_VALUE='IBM'", …)
SFS View	-
DB View	CREATE VIEW MY_PROCESS_LIST (PIID, ID) AS SELECT PI.PIID, (SELECT PA.VALUE FROM PROCESS_ATTRIBUTE PA WHERE PA.NAME='ID' AND PA.PIID=PI.PIID) FROM PROCESS_INSTANCE PI, QUERY_PROPERTY QP WHERE QP.NAME='Customer' AND QP.STRING_VALUE='IBM' AND QP.PIID=PI.PIID
BFM Query against DB View as custom table	bfm.query("MY_PROCESS_LIST.PIID, MY_PROCESS_LIST.ID")
Custom Table Definition	<?xml version="1.0" encoding="UTF-8"?> <customtable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.ibm.com/schemas/workflow/wswf/customtable" xsi:schemaLocation="http://www.ibm.com/schemas/workflow/wswf/customtable http://www.ibm.com/schemas/workflow/wswf/customtable"> <querytableinfo tablename="MY_PROCESS_LIST" aliasname=" MPL" joinlevel="3"> <joincolumn column="PIID" /> <joincolumn column="PIID" target="WORK_ITEM" /> <querycolumninfo columnname="PIID" type="TYPE_ID" isNullable="false" /> <querycolumninfo columnname="ID" type="TYPE_STRING" isNullable="false" /> </querytableinfo> </customtable>