SQL Execution Plan View Design

V0.1

Change History:

Date	Version	Updated by	Comments
2005/12/26	0.1	Dafan Yang	Initial version
2006/01/05	0.2	Dafan Yang	Updated according internal reviewers' feedback

Contents

1. INTRODUCTION 1

1.1 SIMPLE INTRODUCTION 1
1.2 PURPOSE 1
1.3 OVERVIEW 1
1.4 SCOPE 1
1.5 ACRONYMS AND ABBREVIATIONS 1
1.6 GLOSSARY 1
1.7 REFERENCES 2

2. FUNCTIONALITY 2

2.1 REQUIREMENTS 2
2.1.1 Functional Requirements 2
2.1.2 Exclusions (Requirements not considered or explicitly excluded) 2
2.1.3 System Interface Requirements 2
2.1.4 Compatibility/Migration Requirements 3
2.1.5 Security Requirements 3
2.1.6 Repository Requirements 3
2.1.7 Deployment Requirements 3
2.1.8 Porting Implications 3
2.1.9 Standards Compliance 3
2.1.10 Documentation Requirements 3
2.1.11 Internationalization/Localization Requirements 3
2.1.12 Section 508 Requirements 3
2.1.13 Testing Requirements 3
2.2 USE CASES 3
2.2.1 [HLUC-01] Supports displaying text execution plan 3
2.2.2 [HLUC-02] Supports displaying tree-structure graphic execution plan in EPV 4
2.2.3 [HLUC-03] Provides generic and extensible preference page 5
2.2.4 [HLUC-04] Keeps execution plan history for exporting purpose 6
2.2.5 [HLUC-05] Supports importing execution plans 7
2.3 REQUIREMENTS TRACEABILITY MATRIX 8
2.4 USER INTERFACE 8
2.4.1 Text execution plan 8
2.4.2 Graphic execution plan 8
2.4.3 Multiple steps graphic execution plan 9
2.4.4 Preference page for EPV 9
2.4.5 Plan history for exporting 11
2.4.6 Menu items on view action bar 11
2.4.7 Relationships between UI items and Class fields 12

3. DESIGN 13

3.1 FRAMEWORK 13
3.1.1 Model & Core interfaces/classes 13
3.1.2 UI classes 14
3.1.3 Tree-structure graphic execution plan 15
3.1.4 APIs for consumer 15
3.1.4.1 PlanRequest 15
3.1.4.2 EPVFacade 16
3.1.4.3 IExecutionPlanDocument 17
3.1.4.4 IPlanDrawer 17
3.1.4.5 AbstractPlanDrawer 18
3.1.4.6 IPlanParser 18
3.1.4.7 IPlanService 19
3.1.4.8 TreePlanNodeComponent 19
3.1.4.9 TreePlanNodeComposite 22
3.1.4.10 TreePlanNodeLeaf 22
3.1.4.11 TreeExecutionPlanDocument 23
3.1.4.12 TreePlanDrawer 24
3.2 PLUGIN 24
3.2.1 Extension Points 24
3.2.2 Extensions 24
3.2.3 Helper Classes 24
3.2.4 Dependencies 24
4. TUTORIAL: HOW TO USE EPV 25
5. REVIEW COMMENTS/PENDING ISSUES/ACTION ITEMS 25

1. Introduction

1.1 Simple Introduction

SQL Execution Plan View (EPV) is a component in SQL Development Tools, which is a subproject of the Eclipse Data Tools (DTP) project; it is a view to show SQL query execution plans, which are instructions defined by data server to execute queries.

There are two kinds of execution plan altogether: text execution plan and graphic execution plan. Text execution plan is pure text describing the execution plan of a query or stored procedure against a database server; compared with text execution plan, graphic execution plan is intuitionistic and easy to understand. For details about execution plan itself, please refer to related vendor's technical documents.

We will use text widget to display text execution plan and use draw2d to draw graphic execution plan on EPV.

1.2 Purpose

This document specifies the funcational requirements, user interface, framework and other design issues of Execution Plan View.

1.3 Overview

The following contents are addressed in this document:

Requirements

Use cases

User interface

Framework

System interface

Tutorial

1.4 Scope

SQL Execution Plan View is to display SQL execution plans. The execution plan retrieval mechanism is outside the framework's scope and should be implemented by vendor-specific plug-ins.

1.5 Acronyms and Abbreviations

Abbreviation/ Acronym	Term or Description
DTP	Eclipse Data Tooling Platform
EPV	SQL Execution Plan View

1.6 Glossary

Term	Definition
Step	A step is a sub-execution plan in a multiple steps execution plan
Execution Plan	SQL Execution Plan
Text execution plan	Use pure text to describe the execution plan, EPV will directly display the execution plan string passed by the consumer.
Graphic execution plan	Use graphic to show the execution plan to end-users. Consumers need to tell EPV how to draw the execution plan.
Tree-structure graphic execution plan	Use tree to discribe the execution plan.

1.7 References

Date	Document	Author
	<Link>	

2. Functionality

2.1 Requirements

2.1.1 Functional Requirements

[REQ 001] Supports displaying text execution plan

[REQ 002] Supports displaying tree-structure graphic execution plan

[REQ 003] Provides generic and extensible preference page

[REQ 004] Keeps execution plan history for exporting purpose

[REQ 005] Supports importing execution plans

2.1.2 Exclusions (Requirements not considered or explicitly excluded)

[REQ 006] The execution plan retrieval mechanism is outside of this framwork's scope

2.1.3 System Interface Requirements

[REQ 007] Neither the API consumers nor the EPV depend on DTP SQL Editor Framework.

2.1.4 Compatibility/Migration Requirements

N/A

2.1.5 Security Requirements

N/A

2.1.6 Repository Requirements

N/A

2.1.7 Deployment Requirements

N/A

2.1.8 Porting Implications

N/A

2.1.9 Standards Compliance

N/A

2.1.10 Documentation Requirements

[REQ 008] Tooltip should be provided

2.1.11 Internationalization/Localization Requirements

[REQ 009] Support all encoding which operation system supports when exporting plans

2.1.12 Section 508 Requirements

[REQ 010] Should satisfy section 508 requirements

2.1.13 Testing Requirements

N/A

2.2 Use cases

Notice that one high-level use case contains at least one detailed use cases. (HLUC --- high level use case). In this section, we may have two kinds of users --- consumers of EPV, end users.

2.2.1 [HLUC-01] Supports displaying text execution plan

The detailed use cases of HLUC-01 are listed as follows:

USE CASE #1	Display text execution plan in EPV

User	Consumers of EPV	
Goal in Context	The consumer can display the text execution plan retrieved from the data server onto the EPV.	
Scope & Level	EPV	
Preconditions	The text execution plan is generated	
Success End Condition	The execution plan is successfully displayed	
Failed End Condition	Error occurs when displaying the execution plan	
Primary, Secondary Actors	The user, the Eclipse platform, the consumer, EPV	
Trigger	The consumer calls related method to display text execution plan on EPV	
DESCRIPTION	**Step**	**Action**
	1	The consumer calls the related method to display text execution plan on EPV.
	2	The text execution plan is displayed.
	3	
	4	
	5	
EXTENSIONS	**Step**	**Branching Action**
	2a	If error occurs, the error message will also be displayed on EPV
SUB-VARIATIONS		**Branching Action**

2.2.2 [HLUC-02] Supports displaying tree-structure graphic execution plan in EPV

In fact, EPV supports any kinds of graphic execution plan, but we only implements tree-structure graphic execution plan, to display other shape of graphic execution plan, the consumer need to implement some interfaces, please refer to section 4 for details.

The detailed use cases of HLUC-02 are listed as follows:

USE CASE #2	Display tree-structure graphic execution plan in EPV
User	Consumers of EPV

Goal in Context	The consumer can display the tree-structure graphic execution plan retrieved from the data server onto the EPV.	
Scope & Level	EPV	
Preconditions	The tree-structure graphic execution plan is generated	
Success End Condition	The execution plan is successfully displayed	
Failed End Condition	Error occurs when displaying the execution plan	
Primary, Secondary Actors	The user, the Eclipse platform, the consumer, EPV	
Trigger	The consumer calls related method to display the tree-structure execution plan on EPV	
DESCRIPTION	**Step**	**Action**
	1	The consumer calls the related method to display the tree-structure execution plan on EPV.
	2	The graphic execution plan is displayed.
	3	
	4	
	5	
EXTENSIONS	**Step**	**Branching Action**
	2a	If error occurs, the error message will also be displayed on EPV
SUB-VARIATIONS		**Branching Action**

2.2.3 [HLUC-03] Provides generic and extensible preference page

Every vendor can have its own options for execution plan, this can be done through implementing an interface, please refer to section 4 for details.

The detailed use cases of HLUC-03 are listed below:

USE CASE #3	Provides generic and extensible preference page
User	Consumers of EPV
Goal in Context	The consumer can extend our preference page for execution plan to provide its own options.
Scope & Level	EPV

Preconditions	N/A	
Success End Condition	N/A	
Failed End Condition	N/A	
Primary, Secondary Actors	The eclipse platform, the consumer, EPV, the preference page	
Trigger	N/A	
DESCRIPTION	**Step**	**Action**
	1	
	2	
	3	
EXTENSIONS	**Step**	**Branching Action**
SUB-VARIATIONS		**Branching Action**

2.2.4 [HLUC-04] Keeps execution plan history for exporting purpose

The detailed use cases of HLUC-04 are listed as follows:

USE CASE #4	Keeps execution plan history for saving/exporting purpose	
User	End users	
Goal in Context	Allow the user to export the execution plans	
Scope & Level	EPV	
Preconditions	There are execution plans generated	
Success End Condition	Successfully exporting the plans	
Failed End Condition	Error occurs while exporting	
Primary, Secondary Actors	The user, the eclipse platform, EPV	
Trigger	The user gestures to export the execution plans	
DESCRIPTION	**Step**	**Action**
	1	The user gestures to export the execution plans

	2	The execution plans are successfully exported
	3	
EXTENSIONS	**Step**	**Branching Action**
	2a	If exporting fails, a error message dialog will pop up to report the error information
SUB-VARIATIONS		**Branching Action**

2.2.5 [HLUC-05] Supports importing execution plans

The detailed use cases of HLUC-05 are listed as follows:

USE CASE #8	Import the saved execution plans	
User	End users	
Goal in Context	Allow the user to load the exported execution plans	
Scope & Level	EPV	
Preconditions	There are exported execution plans	
Success End Condition	Successfully importing the plans	
Failed End Condition	Error occurs while importing the plans	
Primary, Secondary Actors	The user, the eclipse platform, EPV	
Trigger	The user gestures to import the execution plans	
DESCRIPTION	**Step**	**Action**
	1	The user gestures to import the execution plans
	2	The execution plans are successfully imported
	3	
EXTENSIONS	**Step**	**Branching Action**
	2a	If importing fails, a error message dialog will pop up to report the error information
SUB-VARIATIONS		**Branching Action**

2.3 Requirements traceability matrix

Use case	Requirements
[HLUC-01]	**[REQ 001] [REQ 006] [REQ 007] [REQ 008] [REQ 010]**
[HLUC-02]	**[REQ 002] [REQ 006] [REQ 007] [REQ 008] [REQ 010]**
[HLUC-03]	**[REQ 003] [REQ 007] [REQ 008] [REQ 010]**
[HLUC-04]	**[REQ 004] [REQ 007] [REQ 008] [REQ 009] [REQ 010]**
[HLUC-05]	**[REQ 005] [REQ 007] [REQ 008] [REQ 010]**

2.4 User Interface

2.4.1 Text execution plan

The SQL statement is displayed on the top of EPV.

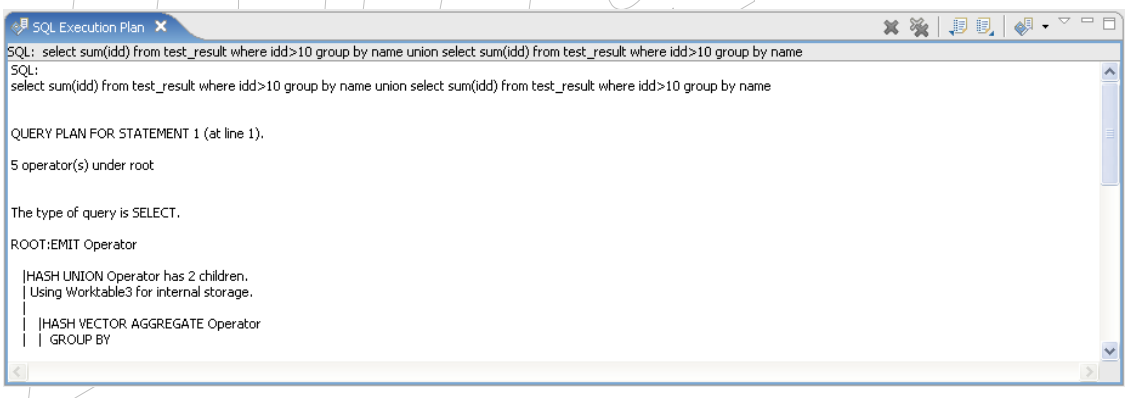


Figure 1

2.4.2 Graphic execution plan

The SQL statement is displayed on the top of EPV. In graphic display mode, the left display area is used to display the graphic plan itself, the right area is used to display some detailed information on a *browser*. The following picture gives an example of tree-structure execution plan, each node has some detailed information.

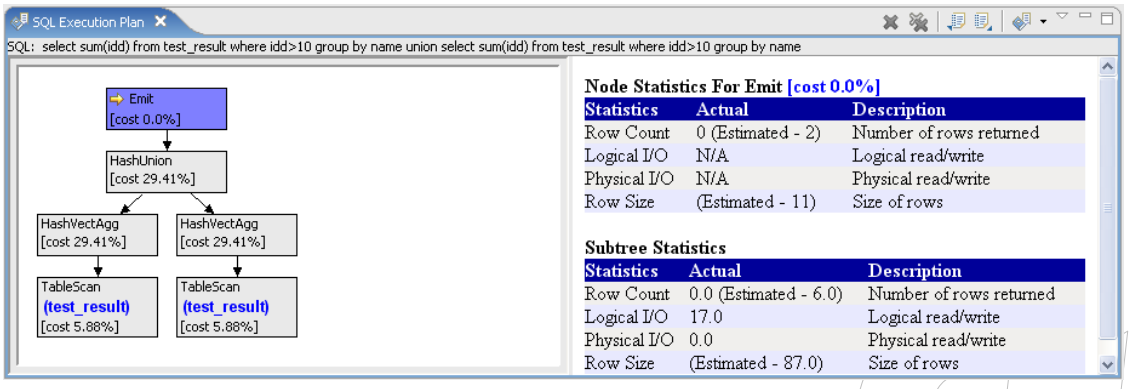


Figure 2

2.4.3 Multiple steps graphic execution plan

For multiple steps graphic execution plan, graphic plan of a SP for instance, we use combo box to list all the steps.

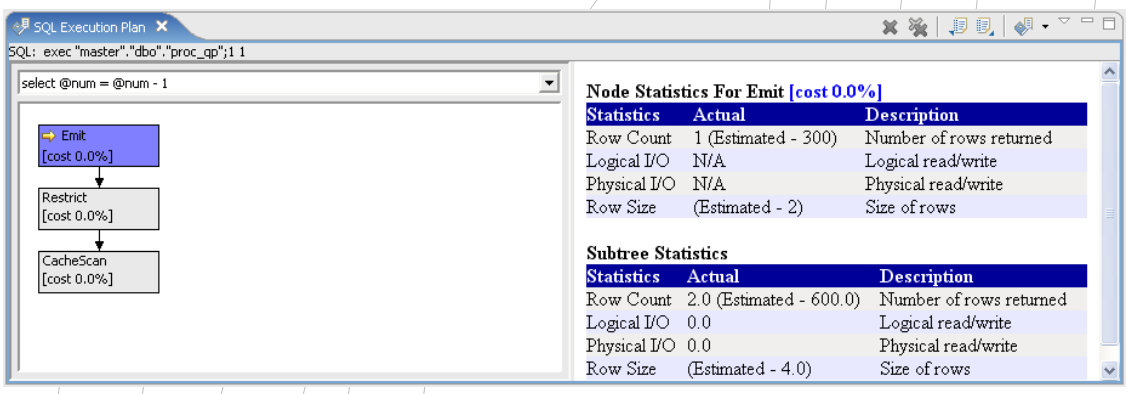


Figure 3

2.4.4 Preference page for EPV

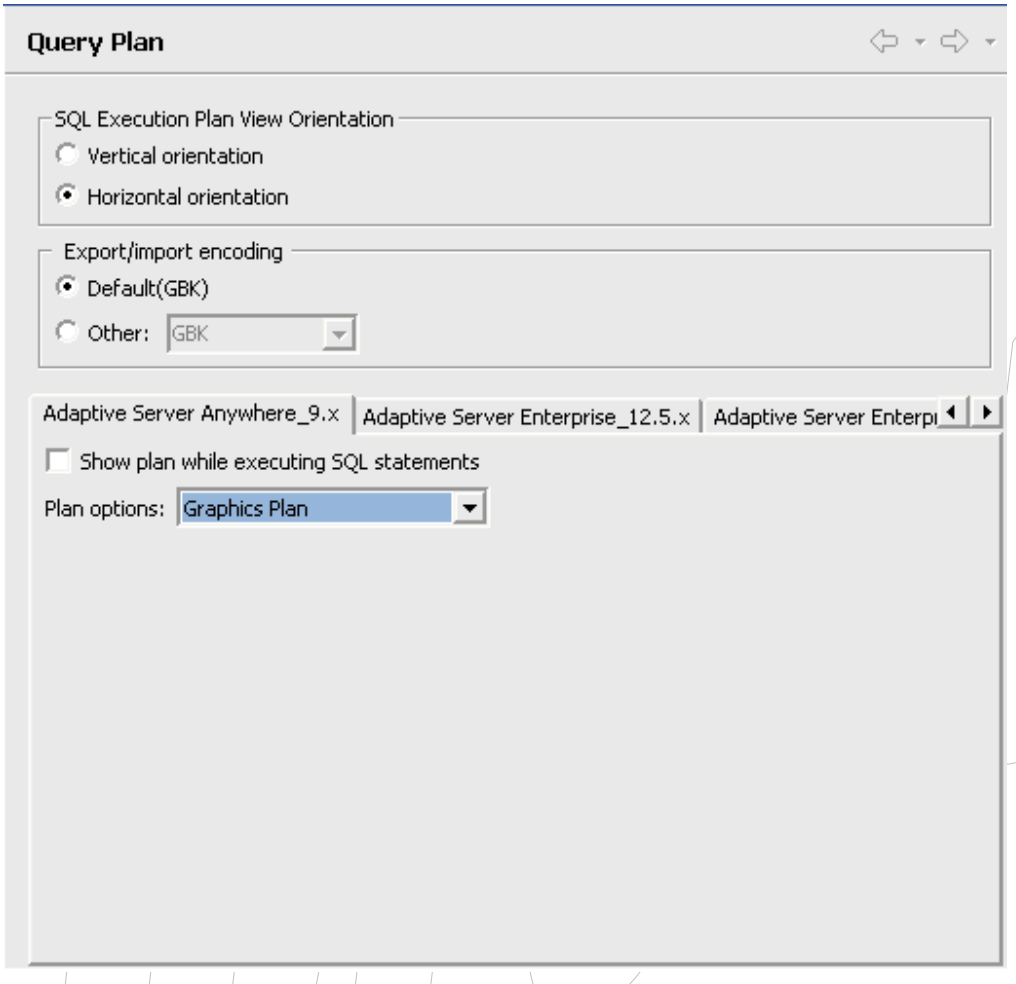


Figure 4

Option Name	**Description**	**Default**	**Enable condition**
Vertical orientation	Make the EPV vertical layout by default	Unselected	N/A
Horizontal orientation	Make the EPV horizontal layout by default	Selected	N/A
Default encoding	Use the JDK default encoding to export/import execution plans	Selected	N/A
Other encoding	Use other encoding to export/import execution plans	Unselected	N/A
Extended vendor options	Vendor specific execution plan options, every vendor is expected to have one tab. (**Some samples are given on this picture --- Adaptive Server Anywhere_9.x …**)		

2.4.5 Plan history for exporting

All the execution plans will be persisted util the Eclipse is shut down.

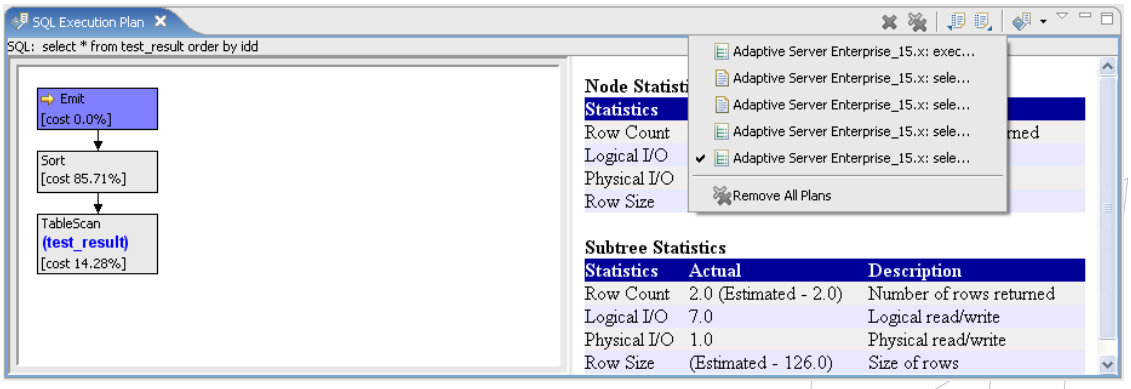


Figure 5

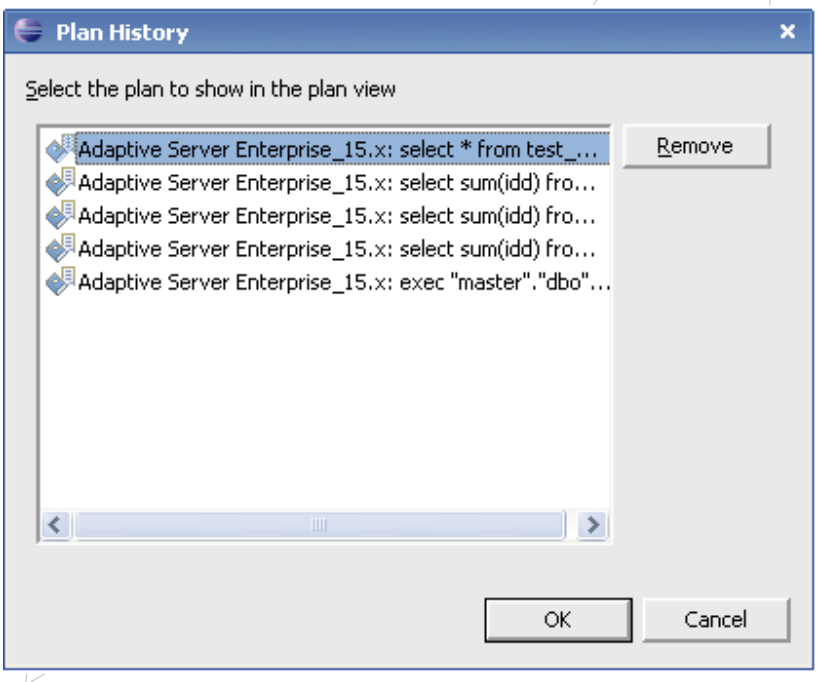
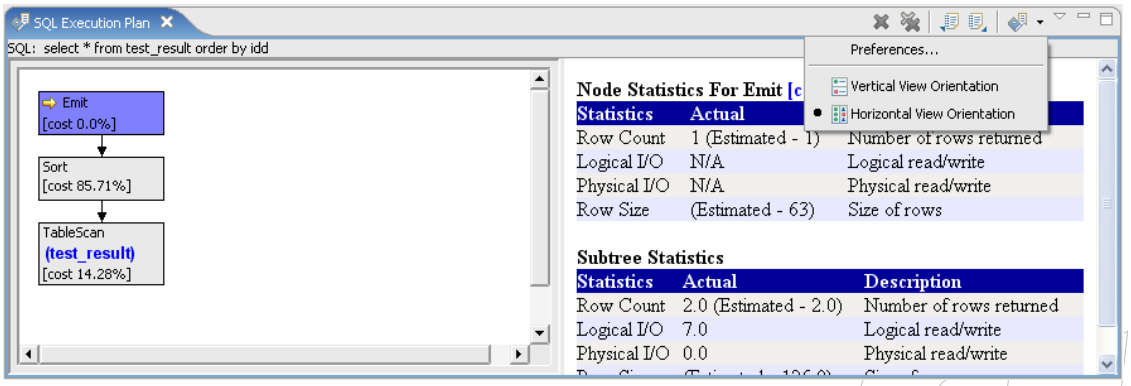


Figure 6

2.4.6 Menu items on view action bar



Name	Description	Default
Preferences…	Open the preference page of EPV	N/A
Vertical View Orientation	Set the orientation of EPV to vertical	Not selected
Horizontal View Orientation	Set the orientation of EPV to horizontal	Selected

2.4.7 Relationships between UI items and Class fields

The following diagram shows the relationships between UI items and fields defined in EPV APIs.

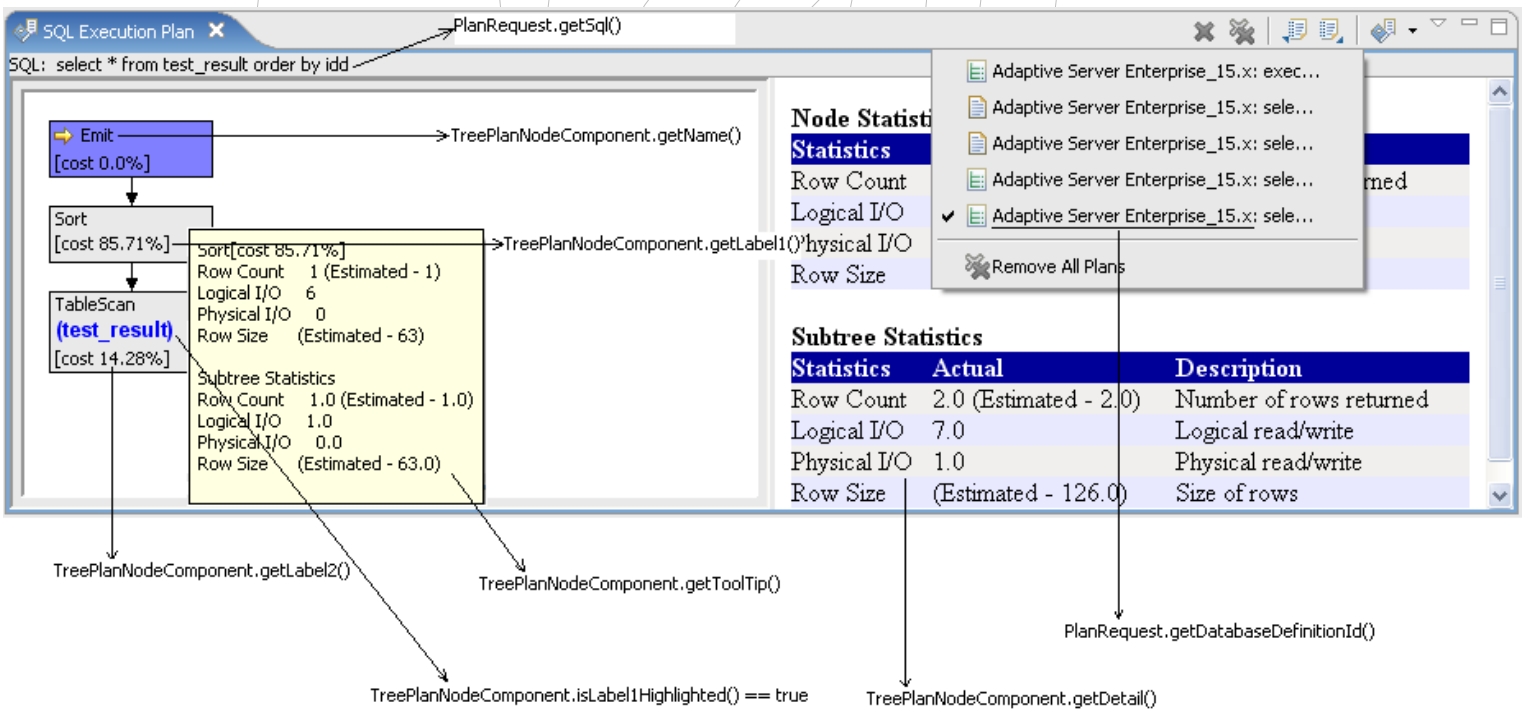


Figure 7

The following diagram shows the name of the sub execution plan.

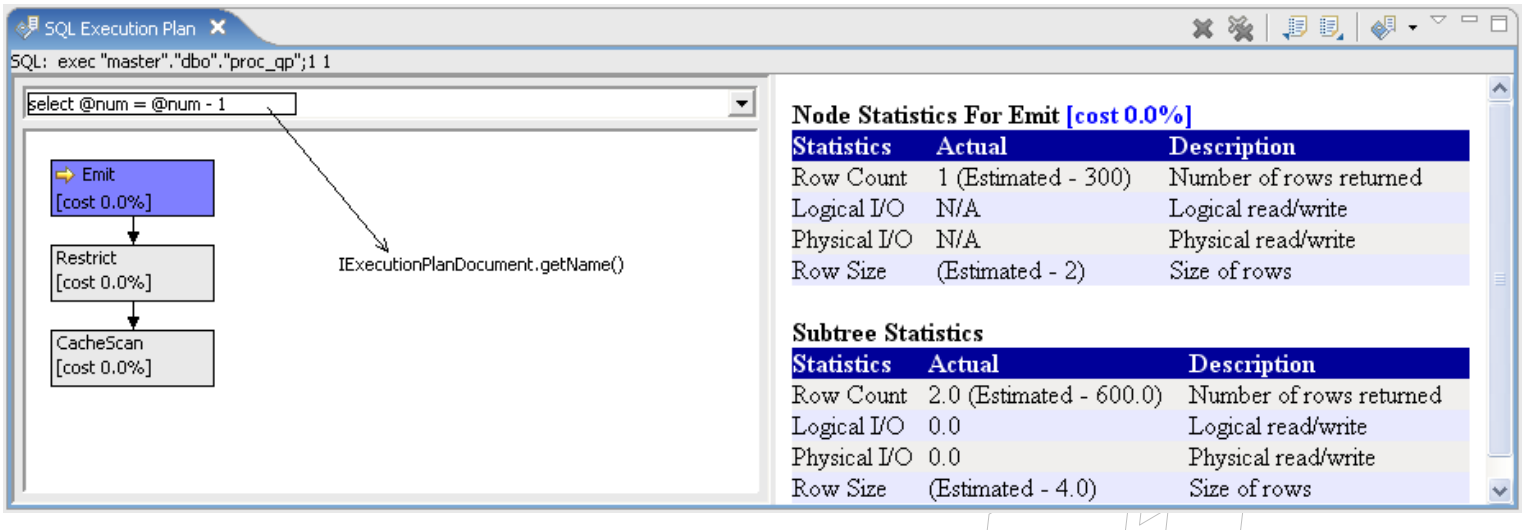
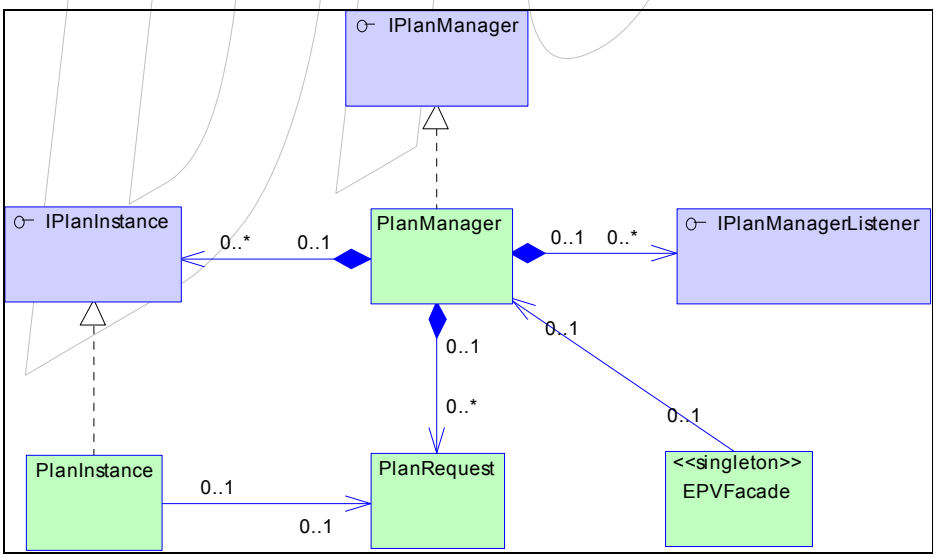


Figure 8

3. Design

3.1 Framework

3.1.1 Model & Core interfaces/classes



Class Diagram 1

PlanRequest: Start point to use EPV, every time when the consumer needs to display an execution plan on EPV, it must construct an instance of *PlanRequest* first, which encapsulates all necessary information.

IPlanInstance: Internally, this is a model interface; every instance of this interface stands for an execution plan, it contains the raw data of the execution plan and the status --- RUNNING, SUCCESS or FAILED.

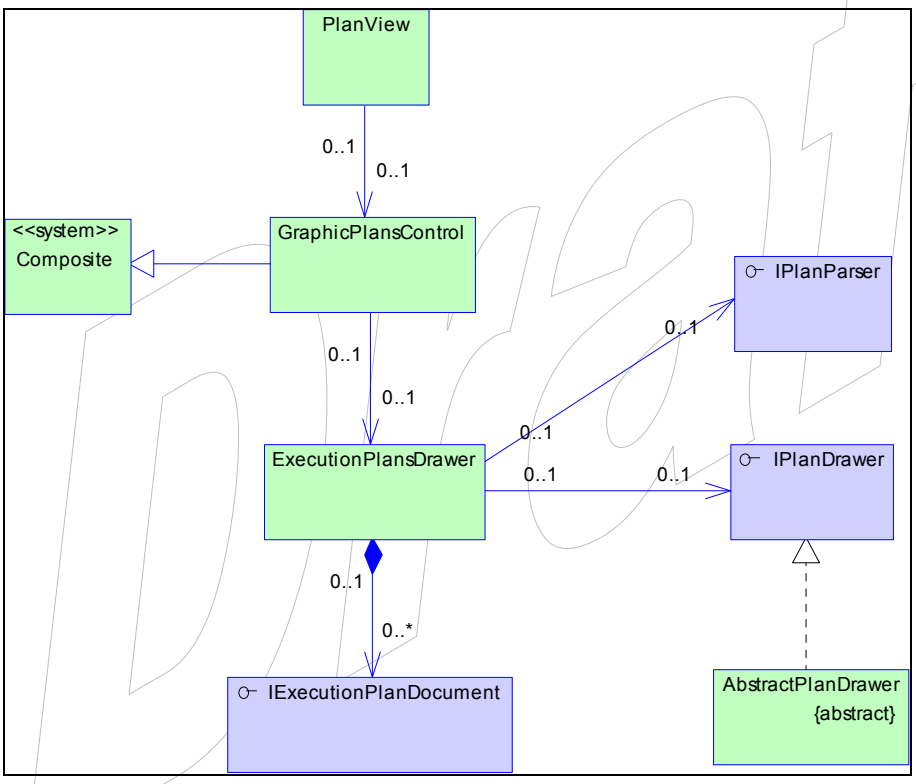
PlanInstance: A standard implementationof *IPlanInstance*.

PlanManager: The control class to manage all the execution plans, it holds a map mapping from instances of *PlanRequest* to those of *IPlanInstance*; it is responsible for notifying all listeners when execution plan events occur (plan is created, removed, etc.).

IPlanManager: Interface for *PlanManager*.

IPlanManagerListener: All instances of this interface registered to *PlanManager* will be notified when execution plan events occur.

3.1.2 UI classes



Class Diagram 2

PlanView: The EPV.

IExecutionPlanDocument: An instance of this interface renders a step in a multiple steps execution plan (For example, the execution plan of a stored procedure). For a normal SQL query statement, there is only one step in the whole execution plan. Instance of this interface can be obtained through *IPlanParser*, *IPlanParser* can parse the raw data of execution plan into instances of *IExecutionPlanDocument*.

GraphicPlansControl: Used to display graphic execution plans. It extends *Composite* to become a standard SWT widget. It has two display areas: one for graphic execution plan and one for the node details.

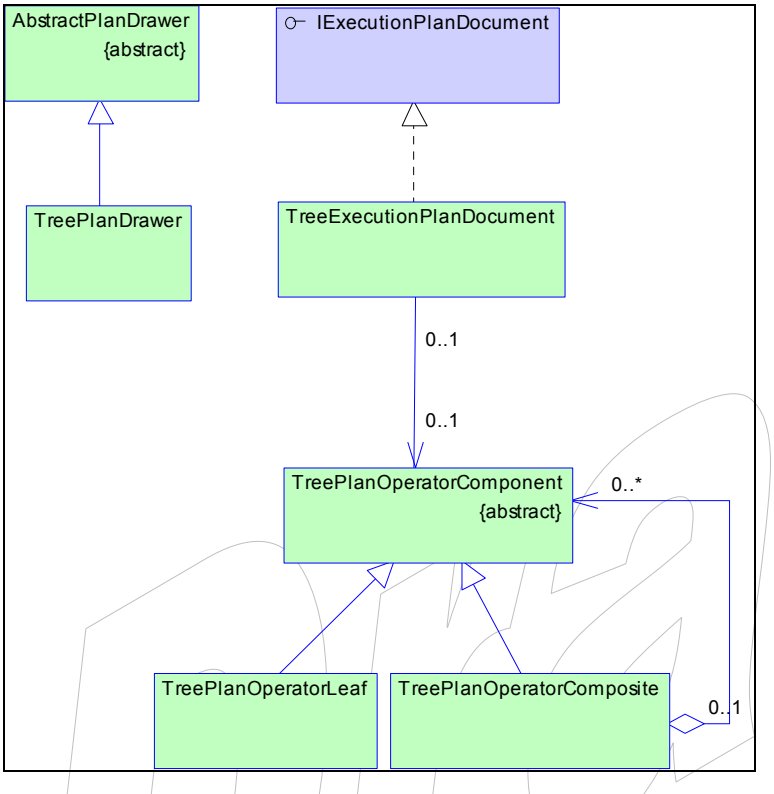
ExecutionPlansDrawer: Used to draw an execution plan, each step is an independent diagram. A combo box is used to list all the sub plans (If there is only one step, the combo box will not be displayed).

IPlanDrawer: A class that implements this interface can draw a graphic execution plan on a *Canvas*.

IPlanParser: Parse the execution plan from row data to modeled data (*IExecutionPlanDocument*).

AbstractPlanDrawer: Vendor can extend this class to provide the execution plan drawer.

3.1.3 Tree-structure graphic execution plan



Class Diagram 3

The *TreeExecutionPlanDocument* is the model class for tree-structure execution plan. *TreePlanDrawer* can draw a tree-structure execution plan on a *Canvas*.

To use tree plan drawer to draw execution plan on EPV, consumer need to parse the execution plan's row data into instances of *TreeExecutionPlanDocument*.

3.1.4 APIs for consumer

We list all the Javadoc of the classes exposed to extenal consumers as following. (Note: section 3.1.4.8 – 3.1.4.12 are API classes for tree-structure graphic execution plan.)

3.1.4.1 PlanRequest

Start point to use EPV (SQL Execution Plan View), every time when the consumer needs to display an execution plan on EPV, it must construct an instance of PlanRequest first. This class encapsulates all necessary information:

- DatabaseDefinitionId: This is used to uniquely identify a database product, the format is: "product name" plus "_" plus "product version", for example, Derby_10.0. This id will be displayed together with the SQL statement as the name of this plan item
- PlanType: Can be text execution plan or graphic execution plan

- SQL: The SQL statement from which the execution is generated

public **PlanRequest**(java.lang.String sql,
java.lang.String databaseDefinitionId,
int planType)

Constructs a plan request

Parameters:

sql - the SQL statement

databaseDefinitionId - the database definition id, use "product_name"_"version" to uniquely identify a database product

planType - the plan type, can be TEXT_PLAN or GRAPHIC_PLAN

Refer figure 7 to see where the defined fields are displayed.

3.1.4.2 EPVFacade

The facade of EPV (SQL Execution Plan View).

public EPVFacade **getInstance**()

Returns the instance of EPVFacade

Returns:

the instance of EPVFacade

---

public boolean **createNewPlanInstance**(PlanRequest request)

Creates an instance of execution plan, to display an execution plan on EPV, consumers must create a new plan instance first

Parameters:

request - the plan request

Returns:

true if creation succeeds, false otherwise

---

public void **planGenerated**(PlanRequest request,
java.lang.Object rawPlan)

Informs the EPV (SQL Execution Plan View) that the plan raw data of the given plan request is successfully generated. The raw data of the execution plan will be set to the plan instance.

Parameters:

request - the plan request

rawPlan - the raw data of execution plan

---

public void **planFailed**(PlanRequest request,
java.lang.Throwable th)

Informs the EPV that error occurs when getting the execution plan from the data server. The error information will be displayed on EPV.

Note that the IPlanParser will be responsible for parsing the raw data of plan into modeled data.

Parameters:

request - the plan request

th - the error information

3.1.4.3 IExecutionPlanDocument

This is the model for drawing purpose. An instance of this interface renders a step (sub execution plan) in a multiple steps execution plan; for example, the execution plan of the following stored procedure has two steps:

```
create procedure test_proc as
begin
 select * from test_table
 select * from test_table1 where id = 5
end
```

For a normal SQL query statement, there is only one step in the whole execution plan. For example, the following SQL statement has only one step:

```
select * from test_table
```

Instance of this interface can be obtained through IPlanParser, IPlanParser can parse the raw data of execution plan into instances of IExecutionPlanDocument

IPlanDrawer can draw instance of this class on a Canvas

public java.lang.String **getName**()

Returns the name of this execution plan documemnt, this name will be displayed in combo box if there are multiple steps in an execution plan.

Returns:

the name of this document, this name will be displayed in combo box if there are multiple steps in an execution plan.

Refer figure 8 to see where "name" field is displayed.

3.1.4.4 IPlanDrawer

Class that implements this interface can draw a sub-execution plan on a Canvas. Vendor can extend AbstractPlanDrawer intead of implementing this interfact from scratch.

public void **drawPlan**(IExecutionPlanDocument document)

Draws the given execution plan

Parameters:

document - the exeuction plan

---

public void **setCanvas**(org.eclipse.swt.widgets.Canvas canvas)

Sets the canvas on which the graphic plan is drawn

Parameters:

canvas - the canvas

---

public void **setBrowser**(org.eclipse.swt.browser.Browser browser)

Sets the browser on which the detail information of one plan node is displayed

Parameters:

browser - the browser

---

public void **init**()

Initializes the drawer, this method will be invoked after the drawer is constructed

3.1.4.5 AbstractPlanDrawer

Consumers of Execution Plan View can extend this class intead of implementing IPlanDrawer from scratch.

public **AbstractPlanDrawer**(org.eclipse.swt.widgets.Canvas canvas,
org.eclipse.swt.browser.Browser browser,
IExecutionPlanDocument query)

Constructor , the default execution plan document is given

Parameters:

canvas - the canvas, will be used to display graphic plan

browser - the browser, will be used to display node detail information

query - the execution plan document

---

public **AbstractPlanDrawer**(org.eclipse.swt.widgets.Canvas canvas,
org.eclipse.swt.browser.Browser browser)

Constructor

Parameters:

canvas - the canvas, will be used to display graphic plan

browser - the browser, will be used to display node detail information

3.1.4.6 IPlanParser

Parse the execution plan from raw data to modeled data --- IExecutionPlanDocument, then IPlanDrawer will draw it on a Canvas

public IExecutionPlanDocument[] **parsePlan**(java.lang.Object rowPlan)

Parses the row data of execution plan to IExecutionPlanDocument array

Parameters:

rowPlan - the row data of execution plan

Returns:

the execution plan documents

3.1.4.7 IPlanService

In order to support graphic execution plan when using EPV (SQL Execution Plan View), consumer must provide the plan parser and plan drawer, which are used to parse the raw plan data and draw graphic plan. If the graphic plan is tree-structure, the consumer can use TreePlanDrawer to draw the plan; in this case, they only need to implement IPlanParser to parse the execution plan into TreeExecutionPlanDocument

public IPlanParser **getPlanParser**()

Returns the plan parser. For one type of database, only one parser is supported.

Returns:

the parser

---

public IPlanDrawer **getPlanDrawer**()

Returns the plan drawer. The plan drawer is responsible for drawing the IExecutionPlanDocument obtained from IPlanParser on canvas.

Returns:

the plan drawer

3.1.4.8 TreePlanNodeComponent

This is the model for tree-structure graphic execution plan. TreePlanDrawer is responsible for drawing a tree-structure graphic plan on a Canvas.

Refer figure 7 to see where the defined fields are displayed.

public **TreePlanNodeComponent**(java.lang.String name,
java.lang.String tip,
java.lang.String detail,
java.lang.Object data,
java.lang.String label1,
java.lang.String label2,
TreePlanNodeComponent parent)

Constructor

Parameters:

name - name of this node

tip - tooltip of this node

detail - detail information of this node

data - data of this node, consumer can put anything

label1 - first label, will be displayed on the node

label2 - second label , will be displayed on the node

parent - parent node of this node

---

public abstract java.util.ArrayList **getChildren**()

Returns the children of this node, return null if this node is a leaf

Returns:

the children of this node

---

public abstract int **getChildrenCount**()

Returns the number of children

Returns:

the number of children

---

public abstract TreePlanNodeComponent **getChild**(int index)

Returns the child at the given index (the index is based on 0)

Parameters:

index - the index

Returns:

the child at the given index

---

public abstract void **addChild**(TreePlanNodeComponent child)

Adds child to this node, simply return if this node is a leaf

Parameters:

child - the child

---

public java.lang.String **getDetail**()

Returns the detail information of this node

Returns:

the detail information

---

public void **setDetail**(java.lang.String detail)

Sets the detail information of this node

Parameters:

detail - the detail information

---

public java.lang.String **getName**()

Returns the name of this node

Returns:

the name of this node

---

public void **setName**(java.lang.String name)

Sets the name of this node

Parameters:

name - the name

---

public TreePlanNodeComponent **getParent**()

Returns the parent of this node

Returns:

the parent of this node

---

public void **setParent**(TreePlanNodeComponent parent)

Sets the parent of this node

Parameters:

parent - the new parent

---

public java.lang.String **getToolTip**()

Returns the tool tip of this node

Returns:

the tool tip

---

public void **setToolTip**(java.lang.String tip)

Sets the tool tip of this node

Parameters:

tip - the tool tip

---

public java.lang.Object **getData**()

Returns the data of this node

Returns:

the data of this node

---

public void **setData**(java.lang.Object data)

Sets the data of this node

Parameters:

data - the data

---

public java.lang.String **getLabel1**()

Returns the first label of this node

Returns:

the first label of this node

---

public void **setLabel1**(java.lang.String label1)

Sets the first label of this node

Parameters:

label1 - the label

---

public java.lang.String **getLabel2**()

Returns the second label of this node

Returns:

the second label of this node

---

public void **setLabel2**(java.lang.String label2)

Sets the second label of this node

Parameters:

label2 - the label

---

public boolean **isLabel1Highlighted**()

Checks if should highlight label 1

Returns:

true if label 1 should be highlighted

---

public void **setLabel1Highlighted**(boolean label1Highlighted)

Sets _isLabel1Highlighted

Parameters:

label1Highlighted - the new value

---

public boolean **isLabel2Highlighted**()

Checks if should highlight label 2

Returns:

true if label 2 should be highlighted

---

public void **setLabel2Highlighted**(boolean label2Highlighted)

Sets _isLabel2Highlighted

Parameters:

label2Highlighted - the new value

3.1.4.9 TreePlanNodeComposite

See Javadoc of *TreePlanNodeComponent*.

3.1.4.10 TreePlanNodeLeaf

See Javadoc of *TreePlanNodeComponent*

3.1.4.11 TreeExecutionPlanDocument

public **TreeExecutionPlanDocument**(TreePlanNodeComponent rootNode,
java.lang.String name,
java.lang.Object data)

Constructor

Parameters:

rootNode - the root node

name - the name for this document

data - the arbitray data

---

public java.lang.Object **getData**()

Returns the data of this plan document

Returns:

the data of this plan document

---

public void **setData**(java.lang.Object data)

Sets the data of this plan document

Parameters:

data - the data

---

public TreePlanNodeComponent **getRootNode**()

Returns the root node

Returns:

the root node

---

public void **setRootNode**(TreePlanNodeComponent rootNode)

Sets the root node for this execution plan document

Parameters:

rootNode - the root node

---

public java.lang.String **getName**()

Returns the name of this document

Specified by:

getName in interface org.eclipse.datatools.sqltools.plan.IExecutionPlanDocument

Returns:

the name of this document

---

public void **setName**(java.lang.String name)

Sets the name for this document

Parameters:

name - the name for this document

3.1.4.12 TreePlanDrawer

See Javadoc of IPlanDrawer

3.2 Plugin

ID: org.eclipse.datatools.sqltools.plan

Name: SQL Execution Plan View Plug-in

Class: org.eclipse.datatools.sqltools.plan.internal.PlanViewPlugin

3.2.1 Extension Points

In order to support graphic execution plan when using EPV, consumer must provide the plan parser and plan drawer, which are used to parse the row plan data and draw graphic plan. If the graphic plan is tree-structure, the consumer can use *TreePlanDrawer* to draw plan, in this case, they must implement *IPlanParser* to parse the execution plan into *TreeExecutionPlanDocument*.

```
<extension-point id="planService" name="SQL Execution Plan Service"
schema="schema/planService.exsd"/>
```

To support graphic plan for a specific database, the following information should be given:

databaseVendorDefinitionId: The database vendor definition id, it is used to uniquely identify a database product, the format is: "product name" plus "_" plus "product version", for example, Derby_10.0

serviceClass: A class that implements *IPlanService*.

3.2.2 Extensions

Name	Extension Point	Class	Description
SQL Execution Plan View	org.eclipse.ui.views	org.eclipse.datatools.sqltools.plan.internal.ui.view.PlanView	
Execution Plan Preference Page	org.eclipse.ui.preferencePage	org.eclipse.datatools.sqltools.plan.internal.preference.ExecutionPlanPage	

3.2.3 Helper Classes

N/A

3.2.4 Dependencies

N/A

4. Tutorial: How to use EPV

To be done later with the code contribution.

5. Review Comments/Pending Issues/Action Items

To be appended…