Picasso Database Query Optimizer Visualizer

Version 2.1

Database Systems Lab

Supercomputer Education & Research Centre

and

Department of Computer Science & Automation

Indian Institute of Science, Bangalore, India

February 2011

Dedicated to the IISc Centenary (1909 - 2011)

Picasso Database Query Optimizer Visualizer

DOCUMENTATION INDEX

(version 2.1 - February 2011)

For the latest information, visit the project web-site:
http://dsl.serc.iisc.ernet.in/projects/PICASSO/picasso.html

Download this Documentation in pdf format

Picasso Database Query Optimizer Visualizer

Changes in Version 2.1

In May 2007, we released Version 1 of the Picasso Database Query Optimizer Visualizer, a tool intended to provide a visual metaphor for exploring the intriguing behavior of modern industrial-strength query optimizers on complex queries. The Picasso tool has been received warmly by the database community and is currently operational at several industrial and academic research labs world-wide. The Version 2.0 of Picasso, featuring host of new functionalities was released in Feb 2009 and the change-log can be found here. Version 2.1 of Picasso is mostly a bug-fix release along with one major enhancement of supporting MySql working along with Picasso. Following are the details of the same

New features:

- MySql support: Support for the MySql database engine which was not there earlier, has been added in this version.
- Collation Schemes: When the PSP predicates used in the query template were of the string data type, interpolation of the histogram values was done using the binary sort order (UNICODE order), which was Case sensitive. Now, we have added support for the case insensitive collation scheme (with some restrictions, as specified later).
- Load Packet: In v2.0, only the Save Packet feature was provided, wherein users could save Picasso packets. Now the *Load Packet* feature has been added and the saved packets can be re-visualized.

Bug Fixes:

- Null values in SQL-Server histogram: In our earlier version, if

NULL values were encountered in the histogram of any PSP predicate (as encountered in SQL Server), we would get a java null pointer exception. Due to technical reasons, it has been decided that the PSP predicates should be on non-null attributes, and thus, a Picasso exception will now be issued if the query template contains a PSP predicate which has null values in its histogram.

- 1D diagram rendering: The 1D diagram rendering was faulty in 2.0. This issue has been fixed.
- Reduction on Operator level diagrams: While visualizing plan diagrams at the *Operator* level (instead of the default *Parameter* level), if the user tried to reduce the plan diagram, in case of LiteSeer or CC-Seer reduction algorithms, a java exception was issued and when the reduction algorithm was Cost Greedy, reduction would happen incorrectly. We have now disabled generation of reduced diagrams on the *Operator* level of the plan diagram and allow reduction only on the original plan diagram (*Parameter* level) as the former is not conceptually defined.
- Statistics for Cost Greedy reduction: After reduction using the Cost Greedy Algorithm, the average cost increase was calculated taking all the points into account. We have now modified the average cost increase to take into account only those points which were swallowed. Also, the calculation of maximum cost increase was over estimated and is now corrected to show the accurate number. The statistic about the minimum cost increase has been removed as the quantity is not of much use.
- Reduced Diagram rendering in low video mode: The reduced diagram rendering was buggy in the low video mode. This has been corrected.
- Sanity check in FPC based reduction algorithms: The sanity constants which were used in the FPC based reduction algorithms have been removed and reduction now happens purely on the cost threshold basis.
- IS_SERVER_DEBUG option not getting set: An option was provided in Picasso settings to set the flag IS_SERVER_DEBUG, which when turned on, would print server debug information onto the console. This being a server setting was not handled correctly and hence was not working. This bug has been fixed.
- Scrolling in Legend Panel: If the number of plans was more than a certain number (~150), then the plans were not getting displayed in the legend panel scroll bar. This bug has been fixed by increasing the length of the scroll bar to handle a sufficiently large number of plans (~600).
- SQL parsing: If the SQL query template submitted to Picasso had the

constructs like OUTER/ INNER JOINS, then the mapping between the attributes and their tables were not done correctly. This issue is fixed in this release.

Other Features:

- Limitation in the dimensionality: A limitation on the number of dimensions of a plan diagram is placed to 4, i.e., the maximum number of PSP predicates that can be given in a query template is 4. This restriction is placed because of the tremendous amount of time required to generate even a very low resolution plan diagram in dimensions higher than 5, which make it almost infeasible.
- Handling of new parameters in DB2 plan tree: There were certain new parameters introduced in the plan trees of DB2 9.7, which are now handled to correctly visualize Parameter level plan diagrams.
- DB2 index name in Picasso plan tree: In the visualization of plan trees for DB2, the columns on which the indexes operate were not shown for the index based operators. This has been modified to show necessary details.
- Compiled Plan tree: The visualization of the compiled plan trees has been modified to show leaf level cardinalities pertaining to the relations.

Compatibility with v2.0:

- Diagram Database: Picasso 2.1 is fully compatible with Picasso diagram databases that were created using v2.0.
- JDBC Upgrade: The DB2 JDBC driver supplied in the Picasso distribution has been upgraded to additionally support DB2 9.7.

NEW FEATURES in Version 2.0

Version 2.0 of Picasso has new functionalities that collectively enable users to

a.focus their attention on selected sub-spaces of the diagram domain,
b.reduce the computational and communication overheads of diagram production, and

c. identify robust execution plans that are tolerant of errors in selectivity estimates.

Specifically, the new features include:

Major new features:

- **Custom Resolution for each dimension:** In v1, the diagram resolution had to be the same on all dimensions. Now, users can specify the resolution on a per-dimension basis.
- **Custom Range for each dimension:** In v1, the diagrams were always drawn over the entire selectivity range [0 to 100 percent] in all dimensions. Now, the diagram production can be localized to user-specified sub-ranges along each dimension.
- **Client-side Processing:** In v1, diagram processing operations at the client often incurred significant communication and computational overheads with the server. Now, the diagram packet sent from the server to the client contains sufficient information to support processing almost all operations at the client itself.
- **Approximate Compilation diagram generation:** Producing Picasso diagrams can turn out to be computationally expensive on higher-dimension query templates with fine-grained diagram resolutions. To address this issue, the tool now supports diagram approximation algorithms that explicitly optimize only a subset of the query points in the selectivity space, and *infer* the remaining points, based on error tolerances provided by the user. For practical tolerances, the reduction in computational overheads can be as much as an order of magnitude.
- **Robust Plan Reduction:** The reduction of plan diagrams in v1 was such that the cost-increase-threshold guarantee of plan replacement applied to the optimality region of the replaced plan. The tool now has incorporated efficient algorithms that extend this guarantee to the entire selectivity space. This feature makes it feasible to identify execution plans that are robust to errors in selectivity estimates, a chronic problem in database optimizers.

Additional functionalities:

- **Multi Engine Plan View:** At a given query location, side-by-side view of plans from two different database engines, or the same engine at different optimization levels.

- **Enhanced Plan Legend Panel:** For query templates with three or more dimensions, the plan legend panel for each 2-D slice displays both the global and the slice-specific number of plans. The ordering and coloring of plans in the legend is kept consistent across the slices based on global space coverage.
- **Compressed Diagram Packet:** diagram packets sent from the server to the client are compressed to reduce transfer latency.
- **Multi-core Plan Operators:** operator lists have been extended to support database engines featuring plan operators that are specific to multi-core platforms

Compatibility with v1:

- **Java Environment:** To run Picasso 2.0, the underlying Java platform must be at least JDK6.0 and Java3D1.4.0 (v1 was compatible with JDK1.4.2).
- **Diagram Database:** Picasso 2.0 is fully compatible with Picasso diagram databases that were created using v1.
- **JDBC Upgrade:** The MSSQL JDBC driver supplied in the Picasso distribution has been upgraded to additionally support SQL Server 2008.

Documentation Home

Picasso Database Query Optimizer Visualizer

INTRODUCTION

Welcome to the Picasso database query optimizer visualizer software developed at the Database Systems Lab, Indian Institute of Science, Bangalore, India, by this team.

The Picasso tool, written entirely in Java, is operational on a rich suite of industrial-strength database query optimizers (currently DB2, Oracle, SQL Server, Sybase ASE and PostgreSQL are supported). It is in use at a host of academic and industrial labs world-wide, and can be employed as a

a. query optimizer analysis, debugging, and redesign aid by system developers,
b. query optimization test-bed by database researchers, and
c. query optimizer pedagogical support by database instructors and students.

Given an SQL query template that defines a relational selectivity space and a choice of database engine, the Picasso tool automatically generates a variety of diagrams that characterize the behavior of the engine's optimizer over this relational selectivity space. The diagrams include

1. Plan Diagram: A pictorial enumeration of the execution plan choices.
2. Cost Diagram: A visualization of the associated estimated plan execution costs.
3. Cardinality Diagram: A visualization of the associated estimated query result cardinalities.
4. Reduced Plan Diagram: Shows the extent to which the original plan diagram may be simplified (by replacing some of the plans with their siblings in the plan diagram) without increasing the cost of any individual query by more than a user-specified threshold value.
5. Schematic Plan-tree Diagram: A tree visualization of a selected plan

in the plan diagram.

6. Plan-difference Diagram: Highlights the schematic differences between a selected pair of plans that appear in the plan diagram.
7. Compiled Plan-tree Diagram: A tree visualization of a selected plan at a specific location in the plan diagram, annotated with cost and cardinality information.
8. Foreign Plan-tree Diagram: At a given location in a plan diagram produced on a database engine, a tree visualization of the plan produced by another engine (or the same engine at another optimization level) at this location.
9. Abstract-Plan Diagram: A visualization of the behavior of a selected plan in the plan diagram, when the optimizer is requested to use this specific plan throughout the selectivity space. [This feature is operational only on SQL Server and Sybase ASE.]

Apart from query compilation-related diagrams, Picasso also produces

9. Execution Cost Diagram: A visualization of the runtime query response times.
10. Execution Cardinality Diagram: A visualization of the runtime query result cardinalities.

The name of the tool stems from the observation that many plan diagrams appear similar to cubist paintings – the art genius, Pablo Picasso, was a founding-father of the cubist painting genre.

A history of the Picasso versions, listing the bug fixes and the new functionalities incorporated in each version, is available in Code History.

Documentation Home

Picasso Database Query Optimizer Visualizer

INSTALLATION INSTRUCTIONS

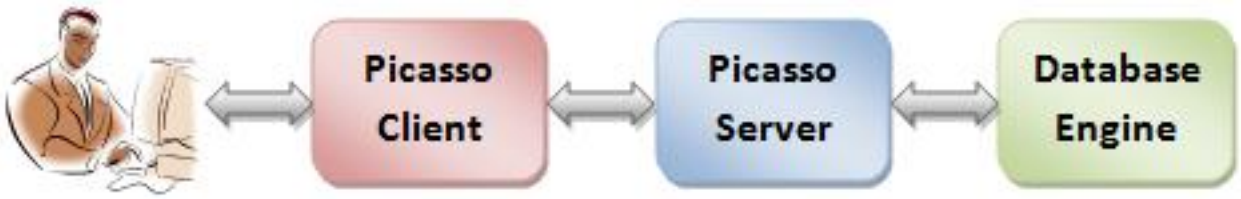
Overview

There are three processes involved in a Picasso setup, as shown in the figure:

(a) the Picasso Client, through which users enter query templates and visualize the associated diagrams;

(b) the Picasso Server, which converts query templates into the equivalent set of query instances, submits these queries to the database engine, and gathers the associated execution plans; and

(c) the Database Engine, which produces efficient execution plans for the queries.



These three processes can all execute on the same machine or on different machines. Further, multiple Picasso clients can connect to a single Picasso server, which in turn can connect to multiple database engines. The client and server machines should support Java compilation and execution, while the client machine should additionally support 3D visualization. A few third-party libraries for visualization and database connection are required for Picasso to function – the details are given in License Information (for

convenience, this support software is included with the Picasso code-base in the ***full*** version).

Picasso is completely written in Java and should, in principle, operate in a platform-independent manner. It has been successfully tested on the following system and database environments:

System platforms:

(a) **Windows 32-bit:** Windows 7 Professional, Intel® Core™2 Duo 2.10GHz, 4 GB RAM, Mobile Intel® 965 Express Chipset Family
(b) **Windows 64-bit:** Windows Vista Business 64-bit, Sun Ultra 24 Intel Core2 QuadCore 3GHz, 8 GB Ram, NVidia Quadro FX 570 graphics card
(c) **Unix 32-bit:** Gentoo Linux (2.6.15 kernel), Pentium-IV 2.4GHz, 1 GB RAM, NVidia Riva TNT2 graphics card
(d)**Unix 64-bit:** Ubuntu Linux (2.6.24 kernel), Sun Ultra 24 Intel Core2 QuadCore 3GHz, 8 GB Ram, NVidia Quadro FX 570 graphics card

Database engines: DB2 8/9, Oracle 9i/10g/11g, SQL Server 2000/2005/2008, Sybase ASE 15, PostgreSQL 8, MySQL 5.1/5.4.1/5.5.9

(For porting to other database engines, please refer to the porting guide.)

Sample Picasso diagrams obtained with the above database engines on the Windows platform are available from the Picasso home page.

Installation Steps

I Setup the Database Engine

1. Install a database engine. Click on the following links for setup information (on

Windows) for specific database engines: DB2 Oracle SQL Server Sybase ASE PostgreSQL MySql. The installation on Unix is on similar lines – please refer the vendor product literature for details.

2. Populate the database with data.
Note: Picasso can be used with generic relational database schemas and SQL query templates. The illustrative examples in the Picasso documentation are with respect to the TPC-H benchmark, and the procedure for setting up this benchmark is given in TPC-H data generation and loading.

3. Then:
a. For DB2 and Oracle, create the ***explain plan*** tables (these tables store the query execution plans generated by these optimizers).
b. For all engines, create ***statistical summaries*** for all relational columns that may be used as Picasso predicates in the query templates. Follow these links for the creation procedure: DB2 Oracle SQL Server Sybase ASE PostgreSQL MySql.

II Download the Picasso Software

1. From Picasso Download, download the Picasso code (version 2.1) – either the ***full*** version, which includes all essential graphics and database libraries, or the ***no-lib*** version, which has only the Picasso source code. Extract its contents on the Server and Client machines. A directory called `Picasso2.1` in which the entire code-base is contained will be created. All paths mentioned in this document and the supporting documentation are with reference to this directory.

2. Read through the documentation given in the PicassoDoc directory.

III Setup the Picasso Server

On the Server machine:

1. Install a Java compiler and execution engine [Sun's JDK 6.0 has been

successfully used in our testbed].

2. If you downloaded the ***full*** version of Picasso, skip to Step 3. On the other hand, if you downloaded the ***no-lib*** version, you need to *manually* add, in the Libraries directory, the necessary database connection libraries for each engine that you wish to have supported in Picasso.

Note: If you use different versions of the database connection libraries, you will need to suitably edit the relevant ***bat*** and ***sh*** files in the PicassoRun\Windows and PicassoRun/Unix directories, respectively.

3. Activate the Picasso interface for the desired database engines.
For Windows, execute ***activatedb.bat*** in the PicassoRun\Windows directory.
For Unix, execute ***activatedb.sh*** in the PicassoRun/Unix directory.

4. Compile the Picasso Server.
For Windows, execute ***compileServer.bat*** in the PicassoRun\Windows directory.
For Unix, execute ***compileServer.sh*** in the PicassoRun/Unix directory.

IV Setup the Picasso Client

On the Client machine:

1. Install a Java compiler and execution engine [Sun JDK 6.0 has been successfully used in our testbed].

2. Install Java3D [Sun Java3D 1.4.0_01 has been successfully used in our testbed]. We recommend the use of the OpenGL version of Java3D wherever possible.

3. If you downloaded the ***full*** version of Picasso, skip to Step 4. On the other hand, if you downloaded the ***no-lib*** version, then you need to *manually* add

the necessary graphics libraries in the Libraries directory.

Note: If you use different versions of the graphics libraries, you will need to suitably edit the relevant ***bat*** and ***sh*** files in the PicassoRun\Windows and PicassoRun/Unix directories, respectively.

4. Compile the Picasso Client.
For Windows, execute ***compileClient.bat*** in the PicassoRun\Windows directory.
For Unix, execute ***compileClient.sh*** in the PicassoRun/Unix directory.

The installation is now complete. To learn how to use Picasso, proceed to the user guide.

Documentation Home

Picasso Database Query Optimizer Visualizer

USAGE GUIDE

Overview

The motivation and conceptual framework underlying the Picasso Database Query Optimizer Visualizer software are presented in this VLDB 2005 paper and this set of presentation slides (Note: The diagram formats and interfaces in these documents are outdated with respect to the current code but the concepts remain essentially the same).

The primary user input to Picasso is a ***query template***. A Picasso query template is an SQL query that additionally features predicates of the form ***"relation.attribute :varies"*** – these attributes are termed as ***Picasso Selectivity Predicates (PSP)***. A sample template (based on Query 14 of the TPC-H benchmark) is shown below, with the two PSPs highlighted in yellow:

```
select    l_extendedprice * (1 - l_discount)
from      lineitem, part
where     l_partkey = p_partkey
          and   l_extendedprice :varies
          and   p_retailprice :varies
```

Each template defines an ***n***-dimensional relational selectivity space, where ***n*** is the number of PSP relations. That is, the selectivity of each of the PSP relations is varied over the range [0-100%], and the objective is to characterize the optimizer behavior over this selectivity space. In the above example, which shows a 2-D template, the selectivity space corresponds to the ***lineitem*** and ***part*** relations.

The query template is converted into a sequence of queries, each of which represents a different point in the selectivity space, through one-sided range predicates of the form ***"relation.attribute ≤ constant"*** implementing each PSP. Note that this formulation also ensures the property of *query subsumption* as we move outwards from the origin of the selectivity space. To estimate the constants that would result in the desired selectivities of the PSP relations, Picasso essentially carries out an "inverse-transform" of the statistical summaries (as present in the database engine's metadata) corresponding to these relations.

To meaningfully cover the full range of selectivities, a Picasso query template should satisfy the

following conditions:

- Each relation can participate in at most one PSP.
- The PSP relations should feature only in join predicates in the query, but not in any other equality or range predicates.
- The permissible data-types for a PSP column are integer, float, string and date (and their equivalents).
- For a PSP column of data-type string, the permitted characters in the string are A-Z, a-z and 0-9.
- The PSP attributes must have pre-generated statistical summaries.
- The PSPs should be on dense-domain attributes in high-cardinality relations.
- The attribute names appearing in the PSPs must either all be unique or disambiguated by explicitly providing REL_NAME.ATTR_NAME in the PSP.

Given a Picasso query template and a choice of database engine, the Picasso tool automatically generates a variety of diagrams that characterize the behavior of the engine's optimizer over this relational selectivity space. The diagrams include:

1. Plan Diagram: A pictorial enumeration of the execution plan choices.
2. Cost Diagram: A visualization of the associated estimated plan execution costs.
3. Cardinality Diagram: A visualization of the associated estimated query result cardinalities.
4. Reduced Plan Diagram: Shows the extent to which the original plan diagram may be simplified (by replacing some of the plans with their siblings in the plan diagram) without increasing the cost of any individual query by more than a user-specified threshold value.
5. Schematic Plan-tree Diagram: A tree visualization of a selected plan in the plan diagram.
6. Plan-difference Diagram: Highlights the schematic differences between a selected pair of plans in the plan diagram.
7. Compiled Plan-tree Diagram: A tree visualization of a selected plan at a specific location in the plan diagram, annotated with cost and cardinality information.
8. Foreign Plan-tree Diagram: At a given location in a plan diagram produced

on a database engine, a tree visualization of the plan produced by another engine (or the same engine at another optimization level) at this location.

9. Abstract-Plan Diagram: A visualization of the behavior of a selected plan in the plan diagram, when the optimizer is requested to use this specific plan throughout the selectivity space. [This feature is operational only on SQL Server and Sybase ASE.]

Apart from query compilation-related diagrams, Picasso also produces:

10. Execution Cost Diagram: A visualization of the runtime query response times.

11. Execution Cardinality Diagram: A visualization of the runtime query result cardinalities.

There are four basic steps in using Picasso:

1. Starting a Picasso Server.

2. Starting the Picasso Client and connecting to the Picasso Server.

3. Connecting to a DB engine through the Picasso Server.

4. Creating (or selecting) a Query Template and generating (or viewing) the associated Picasso diagrams.

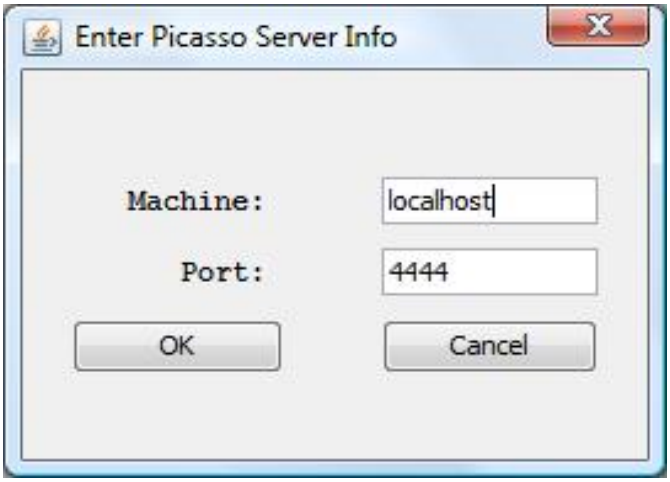
These steps are described in detail in the remainder of this document.

Steps 1–3: Setup

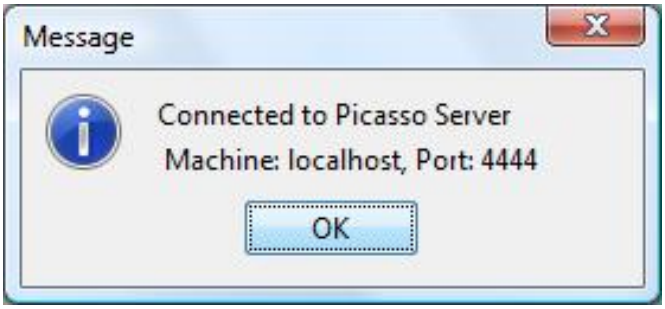
1. Start the Picasso server by executing ***runServer.bat*** | ***runServer.sh*** in PicassoRun\Windows | PicassoRun/Unix directory, giving the port number through which the server will interface with clients as an optional argument (the default port number is 4444). The server will start and run in a console window.

2. Start the Picasso Client by executing ***runClient.bat*** | ***runClient.sh*** in PicassoRun\Windows | PicassoRun/Unix directory. When the client starts, the following 'Welcome' screen appears.

Click on the 'Enter' button in this screen. A dialog asking for the Picasso Server information is displayed with default values.

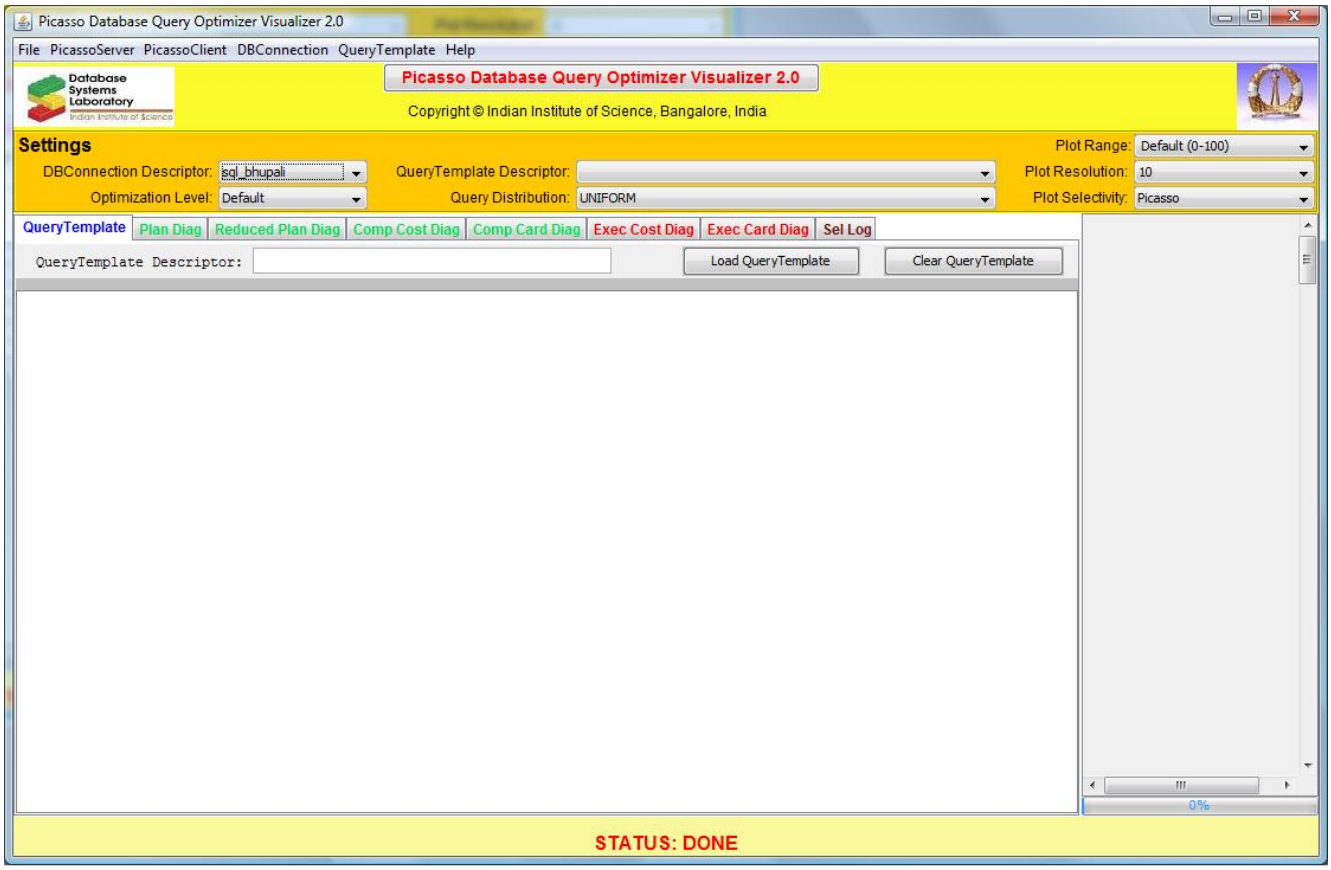


Enter the required information and click ***OK.***



The confirmation message will appear. Click ***OK.***

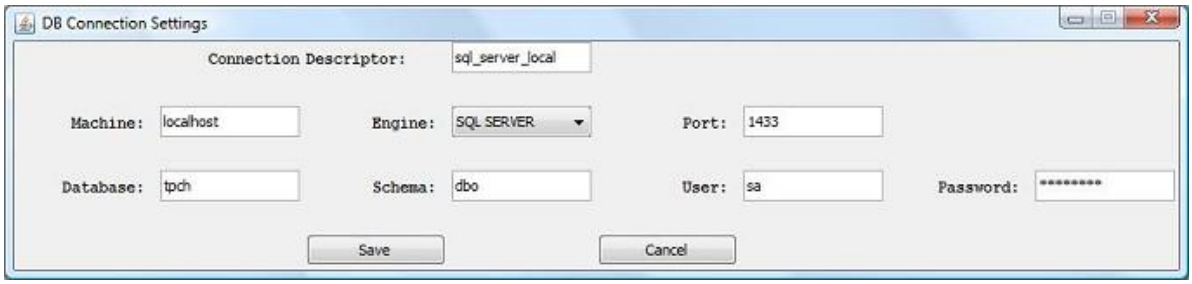
Then the Picasso Client screen appears, looking like:



To learn in detail about the controls on the Picasso Client screen, see Client GUI Controls.

1. In the ***DBConnection*** menu click ***New.*** (Or you can select one of the existing database instances from the ***DBConnection Descriptor*** dropdown list and from the

DBConnection menu click ***Edit.***) The following dialog will appear. Enter the required details and click ***Save***.



Step 4: Generating Picasso Diagrams

2. In the ***Settings*** panel, set the required fields.

3. In the diagram panel, go to the ***QueryTemplate*** tab. Enter a Picasso query template by either typing in a query template or clicking on the ***Load QueryTemplate*** button and selecting a file containing a query template. Then enter a suitable name to identify this template in the ***QueryTemplate Descriptor*** field. (The maximum permitted length of the descriptor is specified by QTNAME_LENGTH in PicassoConstants.java and the default value is 128).
*Note: Representative query templates based on the TPC-H benchmark are available for all the database engines in the **QueryTemplates** directory.*

4. To obtain any of the diagrams (Plan/CompCost/CompCard/ExecCost/ExecCard), click on the associated tabs. If the diagram had been previously generated, the picture is retrieved by the server and shown immediately. Otherwise, a dialog comes up indicating the estimated time to generate and asking whether the diagram should be generated. Click ***OK*** if you want to generate. A new feature of Picasso 2.0 is that an option to generate an *approximate* diagram (as per user-specified error tolerances), instead of the exact diagram, is also provided.

5. As the diagram is being generated, the ***Progress Bar*** (at the bottom of the screen) shows the quantitative progress, while the ***Status Bar*** provides additional details, including the elapsed time and the estimated time to completion. A ***Cancel Processing*** button appears just above the status bar, and clicking this button terminates the on-going generation

process. There is also a ***Pause Processing|Resume Processing*** toggle button that can be used to temporarily suspend and later resume the diagram generation process at the server.

To learn in detail about the semantics of the Picasso Diagrams, see Diagram Semantics.

Command-Line Interface

Apart from the above visual client interface, Picasso also supports a ***command-line interface*** for generating compilation and execution diagrams, which is especially useful for batch processing of query templates. The details are given in Command Line Interface.

User-Specific Files

There are two user-specific files in Picasso: ***DBConnections*** and ***local_conf***, both present in the PicassoRun directory. The former, which is in binary format, is for storing the information about database connections, while the latter, which is in text format, is for customizing the default values of the user-settable constants. These files should be updated only through the controls in the Picasso Client interface, as explained in Client GUI Controls.

Notes:

1. If a Picasso Client is closed, any ongoing process on the Server is not affected. The client can be restarted later and the results of the previous processes viewed in the normal manner.
2. In case of problems, please refer the Trouble-shooting document.

Documentation Home

Picasso Database Query Optimizer Visualizer

CLIENT GUI CONTROLS

This document describes the controls available in the Picasso Client graphical interface. After connection to a Picasso server and DB engine, the whole interface revolves around the concept of a ***Query Template Descriptor (QTD),*** which represents a user-specified Picasso query template augmented with the associated optimizer and diagram-related settings. Within each query template, the attributes chosen for Picasso variation are termed as ***Picasso Selectivity Predicates (PSP)***. The following description is organized based on the control panels that become available as the user moves from top to bottom in the interface:

1. Menu Bar

- ***File*** menu

  Save Screen: Saves a snapshot of the current screen in JPEG format.

  Save Packet: Writes a DiagramPacket object and a Vector of plan-trees into a user-specified file – this will either be a plan diagram packet, a reduced plan diagram packet, or an execution diagram packet, depending on the tab currently displayed. The default name is **"<QTD>_{P|R|E}.pkt"**, with "QTD" corresponding to the query template descriptor, and "P|R|E" corresponding to whether the source is a Plan diagram, Reduced plan diagram or Execution diagram, respectively.
  The user can optionally choose to save the packet in *compressed* (gzipped) format by appropriately setting the "Saved Packet Format" field in the ***Picasso Client →LocalSettings*** menu.
  The saved packets can be read and manipulated in external applications – for example, by including DiagramPacket.java (as well as DataValues.java, TreeNode.java and QueryPacket.java) in the external application, and then writing additional methods to use the data in the packet. A sample program that reads .pkt (and .pkt.gz) files and outputs the values (both diagram and plan tree information) into a human-readable .txt file is available as DisplayDiagramPacket.java in the **Code** sub-directory of **PicassoDoc**. To compile and execute this program, use compileDisplayDiagramPacket.bat and runDisplayDiagramPacket.bat in Windows, and compileDisplayDiagramPacket.sh and runDisplayDiagramPacket.sh in Unix.

  Load Packet: The DiagramPacket and the Vector of plan-trees which are saved using the *Save Packet* feature described above can be loaded back into Picasso using the *Load*

Packet option. It should be noted that the Reduced Diagram and Execution diagrams tabs are invalid when a reduced plan diagram packet is loaded and the Execution diagrams tabs are invalid when a plan diagram packet is loaded. Also, for any plan diagram packet which is loaded, only the Cost Greedy reduction algorithm can be applied.

Print Diagram: Shows a 'Page Setup' dialog and a subsequent 'Print' dialog to print the current diagram. This function is supported only when one of the diagram tabs is selected.

Print Preview: Opens the 'Page Setup' dialog and then pops up a 'Picasso Print Preview' dialog which shows what would be printed. This dialog has a 'Print' button which will open the 'Print' dialog to print the contents of the preview. This function is supported only when one of the diagram tabs has been selected.
Note: Printing in 'Landscape' mode usually produces a better formatted picture as compared to the 'Portrait' mode.

Exit: Closes the Picasso Client.

- ***Picasso Server*** menu
  Connect to PicassoServer: Opens the ***Enter Picasso Server Info*** dialog which asks for the machine name (or IP) and port number of the Picasso Server to which you wish to connect. For connecting to a Picasso server hosted locally on the client machine, it is recommended to use 'localhost' for machine name.

  Server Status: Checks if the Picasso Client is currently connected to any Picasso Server and if so, prints the machine name and port of the server.

- ***Picasso Client*** menu
  Local Settings: Opens a dialog for changing the values of user-settable defaults (defined in PicassoConstants.java) such as the server port, the choice of plan diagram reduction algorithm, etc.

- ***DBConnection*** menu
  Connect: Attempts to connect to the database engine associated with the currently selected ***DBConnection Descriptor*** and reports an error if unable to connect. If the connection is successful, the ***QueryTemplate Descriptor*** drop-down list is populated with the QTDs already available on this engine, if any, for the given database instance.
  For each database connection, all Picasso-related information, including the QTDs and diagrams, is stored in a set of Picasso tables and views within the associated database. The tables are **PicassoQTIDMap, PicassoPlanStore,**

PicassoPlanTree, PicassoPlanTreeArgs, PicassoSelectivityMap, PicassoSelectivityLog, PicassoRangeResMap, PicassoXMLPlan, PicassoApproxMap and the view is **picasso_columns.**

New: Opens the ***DB Connection Settings*** dialog for creating a new ***DBConnection*** instance. The fields in this dialog are:

Connection Descriptor: Specifies the user-defined name that uniquely identifies the DB connection implied by the choice of settings in the dialog.
Machine: Name or IP of the machine on which the database engine is running. (Note: The machine name specified here is from the perspective of the Picasso Server, not the Client. For example, if 'localhost' is specified, it means the machine on which the Picasso Server is running.)
Engine: Select from the drop-down menu the database engine to which you wish to connect (current options are DB2, Oracle, SQL Server, Sybase ASE and PostgreSQL).
Port: Port number of the above database connection. This field is populated with the default value when the ***Engine*** is selected (e.g. selecting DB2 results in port number 50000).
Database: Name of the database instance that you wish to use in the engine (e.g. tpch).
Schema: Name of the schema within the database (e.g. admin).
User: User name with which you can connect to this database instance (e.g. picasso).
Password: Password of the above user.
Save button: Clicking this will close the dialog and save the new connection instance.
Cancel button: Closes the dialog without saving any changes.

Edit: Opens the above dialog, but showing the corresponding values for the connection instance that is selected in the ***DBConnection Descriptor*** drop-down list. You can make any changes and click ***Save*** to save the changes in the existing connection instance.

Delete: Opens the same dialog, but with a ***Delete*** button in place of the ***Save*** button, and with all the fields grayed out. The fields will contain the details of the connection instance which is currently selected in the ***DBConnection Descriptor*** drop-down list. Clicking the ***Delete*** button deletes the connection instance.

Destroy Picasso Database: This control drops (assuming the current user has the requisite permissions) the entire set of Picasso tables and views for the database connection, resulting in the permanent deletion of all associated QTDs.

- ***Query Template*** menu
  Refresh QueryTemplate List: Populates the ***QueryTemplate Descriptor*** list with the QTDs previously defined for the current DB connection.
  Delete QueryTemplate: Deletes the query template which is currently selected in the ***QueryTemplate Descriptor*** list after a confirmation.
  Rename QueryTemplate: Changes the descriptor of the currently selected query template.

- ***Help*** menu
  Documentation Home: Opens the index page of the complete documentation.
  Usage Guide: Opens a help page containing information on how to use the Picasso client.
  Mouse-Key Mappings: Displays a dialog containing the mouse-key mappings for operations on the diagrams.
  About Picasso: Displays a dialog containing information on Picasso including version, home page, contact, etc.

2. Logo Panel

This is found in the title segment of the screen just below the menu bar, and contains three buttons:

- ***Database Systems Laboratory:*** Opens the Database Systems Lab home page in a browser.

- ***Picasso Database Query Optimizer Visualizer:*** Opens the Picasso home page in a browser.

- ***Indian Institute of Science:*** Opens the Indian Institute of Science home page in a browser.

3. Settings Panel

- ***DBConnection Descriptor:*** Drop-down list of connection instances. Selecting any of these instances will result in an attempt to connect to the specified engine and database.

- ***Optimization Level***: Some database engines allow users to choose from different

optimization levels or different optimization goals. For example, DB2 supports a spectrum of numeric settings (0–9) that cover the optimization-quality versus optimization-effort tradeoff, while Oracle provides two choices: minimizing latency (First_Row), or minimizing response time (All_Rows). The optimization levels currently supported for the various engines are as follows, with the default Picasso setting also indicated for the multi-level engines:

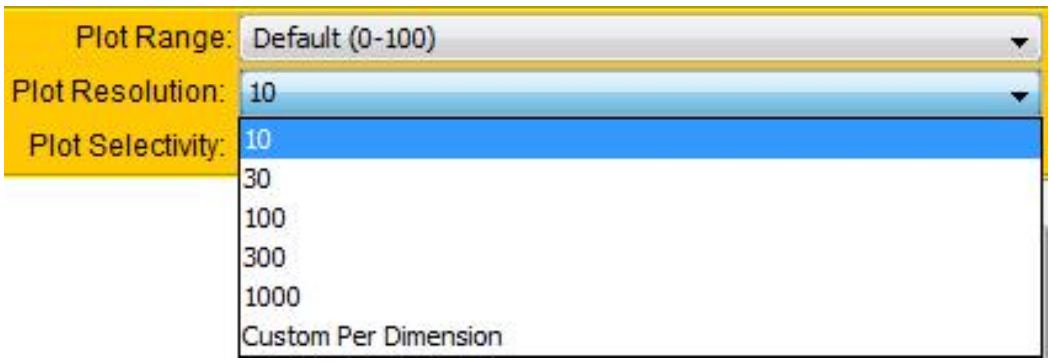
- DB2: *0, 1, 3, 5, 7, 9* (Default = 5)
- Oracle: *First_Row, All_Rows* (Default = All_Rows)
- SQL Server: Installation Default
- Sybase ASE: *allrows_dss, allrows_oltp, allrows_mix* (Default = allrows_mix)
- PostgreSQL: Installation Default

- ***QueryTemplate Descriptor:*** A drop-down list of existing query templates for the database instance referenced by the current DB connection. Depending on whether the diagram corresponding to the query template is Compilation, Execution or Approximate-Compilation, a tag of (C), (C,E) or (A) is shown along with the query template name.

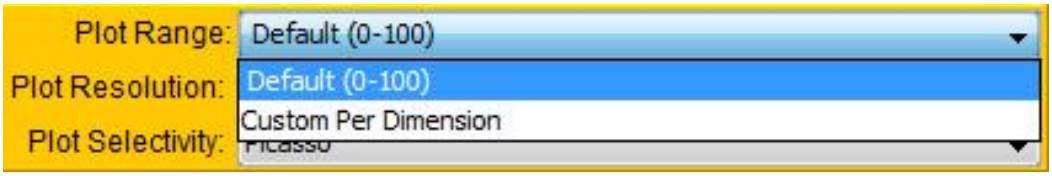
- ***Query Distribution:*** Defines the distribution to be used while generating query points. Currently two distributions are supported: ***Uniform*** and ***Exponential.*** With the Uniform distribution, the query points are evenly located over the selectivity space. On the other hand, with the Exponential distribution, the density of points is maximum near the origin and becomes progressively lower moving outwards in the space. The motivation for the Exponential distribution stems from the observation in the literature that plan density is often high around the origin and along the axes, and it may therefore be useful, from a computational perspective, to focus the query workload on these regions. In the current implementation, the parameters chosen for the exponential distribution produce an *"80-20"* skew – that is, in each dimension, 80 percent of the query points are in the initial 20 percent of the corresponding axis. The default distribution is ***Uniform***.

- ***Plot Resolution:*** Specifies the resolution with which the query points are distributed in each dimension of the selectivity space. For example, if the selected resolution is $\boldsymbol{n_i}$ on each dimension and there are ***d*** dimensions, then the diagram will reflect the output of $\prod_{i=1}^{d} n_i$ distinct queries. The default is to have the same resolution on all dimensions but, by

choosing "Custom per Dimension", the user can specify each dimension to have its own resolution. The available resolutions are 10, 30, 100, 300 and 1000, with the default being 10.



- ***Plot Range:*** The default is for diagrams to be generated over the full range (0-100 percent) of the selectivity space in all dimensions. However, the user can specify, for each dimension, the selectivity range within which the diagram should be generated by selecting "Custom per Dimension". The combination of Custom range and Custom resolution enables users to focus the computational effort on the desired sub-spaces.

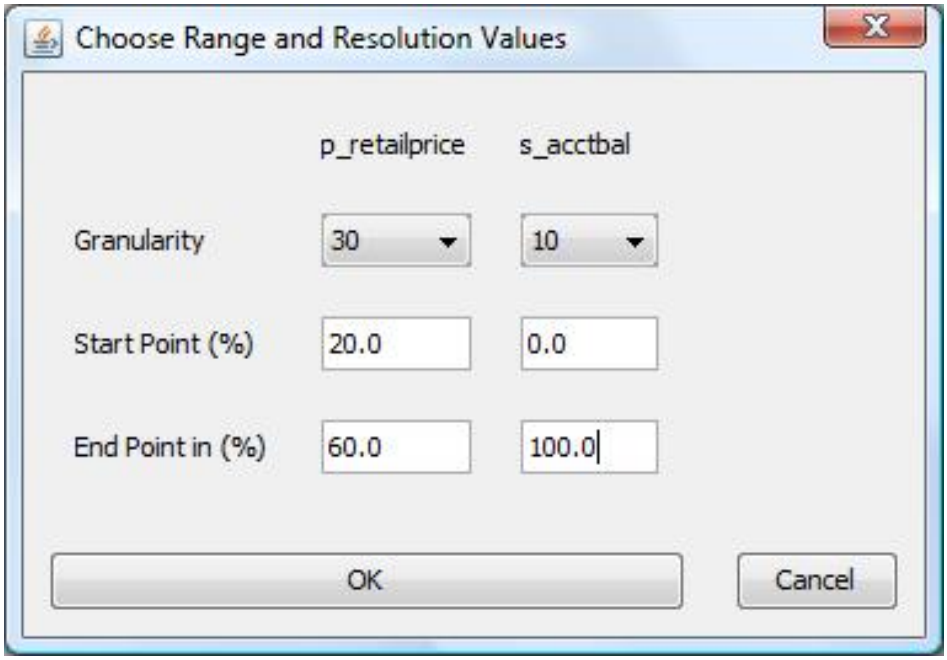


- ***Plot Selectivity:*** The selectivity corresponding to a PSP may sometimes be estimated differently by Picasso and by the optimizer (see Diagram Semantics for a discussion of this issue). This field decides which estimate, ***Picasso*** or ***Engine***, should be used to plot the Picasso diagrams – the default is ***Picasso.***

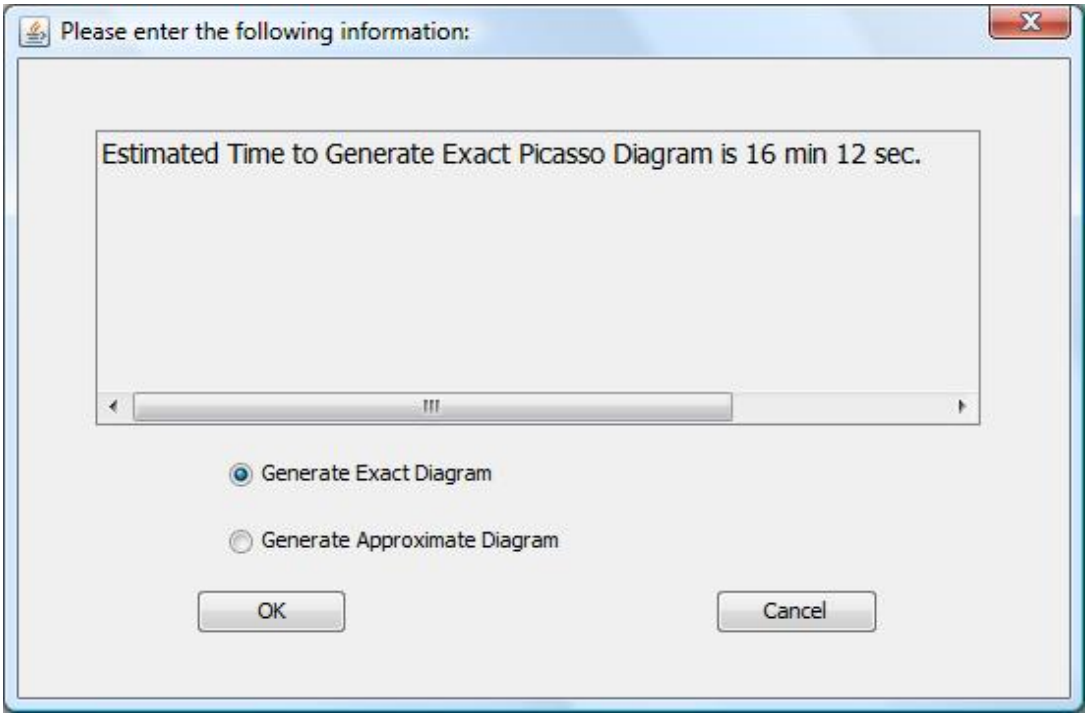
4. Diagram Parameters

There are two pop-up windows which collect the parameter values for the diagram generation.

- ***Range & Resolution:*** This pop-up window appears when the user chooses the Custom per Dimension option in either the *Plot Range* or the *Plot Resolution* field in the Settings Panel.



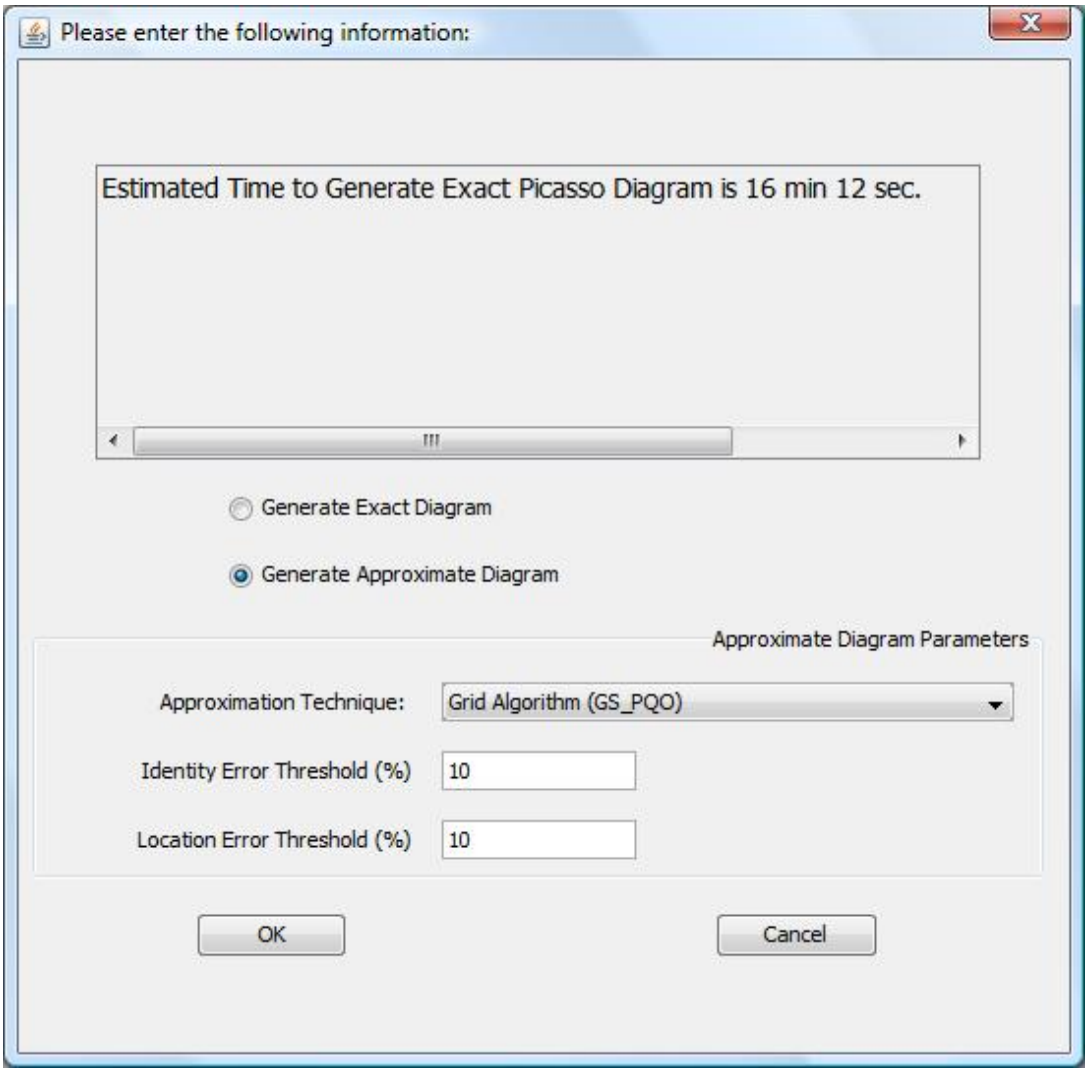
- ***Approximation Parameters:*** This pop-up window displays the estimated time to generate the exact diagram and allows the user the ability to opt for an approximate diagram instead, if so desired.



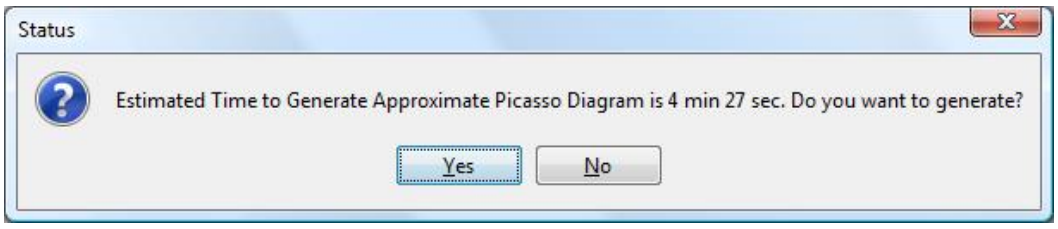
If approximation is chosen, then the following information is collected from the user:

- ***Approximation Technique:*** Specifies choice of approximation algorithm – currently, two choices are available, the sampling-based **RS_NN** and the grid-partitioning-based **GS_PQO** (see Algorithm Details for the details of these various options). The default choice is **GS_PQO**.

- ***Error Thresholds:*** Specifies user's tolerance for *plan-identity* and *plan-location* errors – the first error refers to the percentage of plans that completely fail to appear in the approximate diagram, while the latter refers to the percentage of points in the approximate diagram that have incorrect plan assignments, both errors measured with respect to the exact diagram. The reductions in computational effort due to the approximation are directly impacted by the choices for these tolerances – specifically, the higher the values, the more the savings. The default value is **10%** for both errors since our experience is that with these settings the computational effort is typically reduced by about an order of magnitude as compared to the exact diagram.



When OK is clicked, a new estimate of the generation time corresponding to the approximate diagram is shown.



5. Diagram Bar

- ***QueryTemplate:*** This tab is for entering and viewing the Picasso query template text for a given QTD. As mentioned earlier, a Picasso query template should satisfy the following conditions:

    - Each relation can participate in at most one PSP.
    - The PSP relations should feature only in join predicates in the query, but not in any other equality or range predicates.
    - The permissible data-types for a PSP attribute are int, float, string and date (and their equivalents).
      [Exceptions: Date is not supported in Oracle, and String is not supported in PostgreSQL.]
    - The PSP attributes must have pre-generated statistical summaries.
    - For meaningful coverage of the full range of selectivities, the PSPs should be on dense-domain attributes in high-cardinality relations.
    - The attribute names appearing in the PSPs must either all be unique or disambiguated by explicitly providing RELATION_NAME.ATTRIBUTE_NAME in the PSP.

  *Note: Representative query templates based on the TPC-H benchmark are available for all the supported database engines in the **QueryTemplates** directory.*

The components in this tab are:

QueryTemplate Descriptor (QTD) field: This is a user-defined mnemonic for identifying query templates and would usually also include information from the ***Settings*** panel. The name should be typed in ***before*** generating any of the diagrams. When a new query template is loaded through the ***Load QueryTemplate*** button, or when the ***Clear QueryTemplate*** button is clicked, a default QTD is generated which identifies the current settings, and this descriptor is editable. The default string represents information about the DB Engine, optimization level, query distribution, and plot resolution, all separated by the underscore (_) character. This default string is not generated if the query is typed manually.

Load QueryTemplate button: This opens a file dialog where you can select a query template from among an available set.

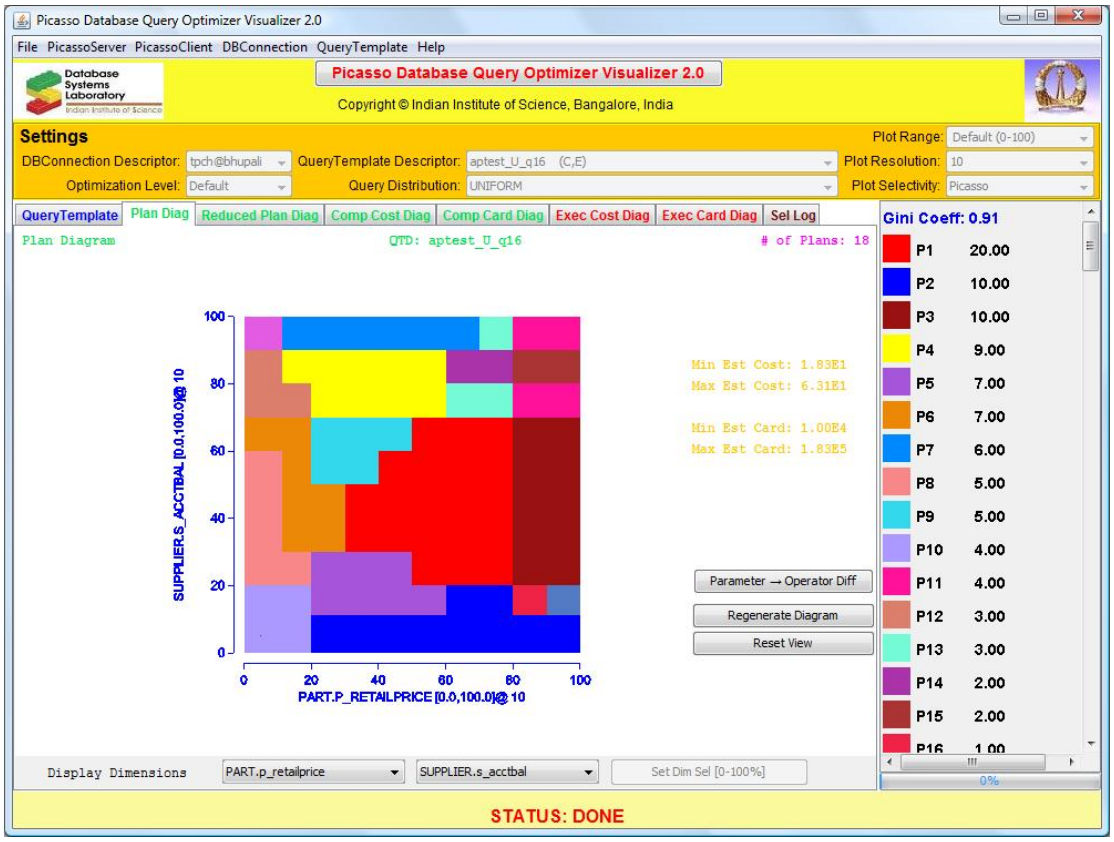
Clear QueryTemplate button: Clears the text area and fills the QTD name according to the settings in the ***Settings*** panel.

Text Area: Type in the query template or load it from a file by clicking on the ***Load QueryTemplate*** button.

Display Dimensions: Select any two of the ***d*** PSP columns in the query template to determine the X (first drop-down list) and Y (second drop-down list) selectivity axes of the initial diagrams to be displayed by the Picasso server. Subsequently, other dimensions can be substituted to obtain different 2-D slices of the ***d***-dimensional hypercube defined by the query template.

The remaining 6 tabs are the diagram tabs. All these diagrams are generated afresh only if they are not from an existing QTD, otherwise the previously generated pictures are shown. Finally, at all times, a fresh generation can be forced by clicking the ***Regenerate Diagram*** button. Note also that for diagrams generated at high resolution (e.g. 1000), both the generation and the loading processes can be time-consuming.

- ***Plan Diag:***



The *Plan Diagram* is a pictorial enumeration of the execution ***plan choices*** made by the optimizer over a relational selectivity space. The plan diagram assigns a unique color to each different optimal plan and assigns this color to all occurrences of the plan in the diagram. The relative percentage space occupied by each plan is also shown in the legend. In this tab, the plan diagram is displayed with the axes labeled with the PSP names and graduated with the selectivity values.

The Plan Diagram is always 2D, except when the query-template has only 1 dimension (i.e one PSP), in which case it is 1D. If the query-template has more than 2 dimensions (i.e. ≥ 2 PSPs), then the plan diagram represents a 2D slice where the ***x*** and ***y*** axes and the constants corresponding to the rest of the dimensions are chosen using the ***Display Dimensions*** and ***Set Dim Sel*** controls of the ***QueryTemplate*** tab.

Set Dim Sel button: This is applicable for query templates with 3 or more

dimensions. It opens a drop-down list for specifying the selectivities of the remaining dimensions, when visualizing 2-D slices of the ***d***-dimensional hypercube defined by the query template.

Diagram Panels (top + right): The QTD is displayed at the center of the top panel, and the number of plans in the current 2-D slice and in the overall diagram are displayed at the right corner. The minimum and maximum estimated values of the costs and cardinalities of the query template are displayed in the middle of the right panel for both the slice and the overall diagram. Further, the selectivities of the remaining dimensions are also shown here. (Note: For 1-D and 2-D query templates, only the overall information is shown.)

The ***"Parameter ↔ Operator Diff"*** toggle button on the right panel is used to toggle the visualization of the plan diagram between

(a) Parameter Diff: plan differentiation on the basis of both operators and their parameters (default setting); and
(b) Operator Diff: plan differentiation solely on the basis of operators

See Diagram Semantics for the technical details on these differentiations.

When any point in the diagram is clicked, it is converted to the closest ***query point*** in the selectivity space. For exact diagrams, these query points are the explicit points at which the query was optimized during the diagram generation process, whereas for the approximate diagrams, they may also be inferred points.

In this tab, you can:

Left-click + drag on the plan diagram to move it around.

Shift + Left-click on the diagram and move the cursor up to zoom in, or down to zoom out.

Right-click at any point in the diagram to view the ***Schematic Plan Tree*** corresponding to the plan choice in that region of the plan diagram.
To view the schematic difference between a pair of plans *right-click* on one plan, drag and drop the mouse on the second plan. This will open a ***Plan Difference*** window showing both the plans, with the differences between the plans highlighted in a color-coded format. (Dragging and dropping the mouse on points corresponding to the *same* plan will simply bring up the corresponding ***Plan Tree***.)

Shift + Right-click on a point in the diagram shows a ***QueryInfo*** message box with information on selectivity values, cost, cardinality, constants and plan number.

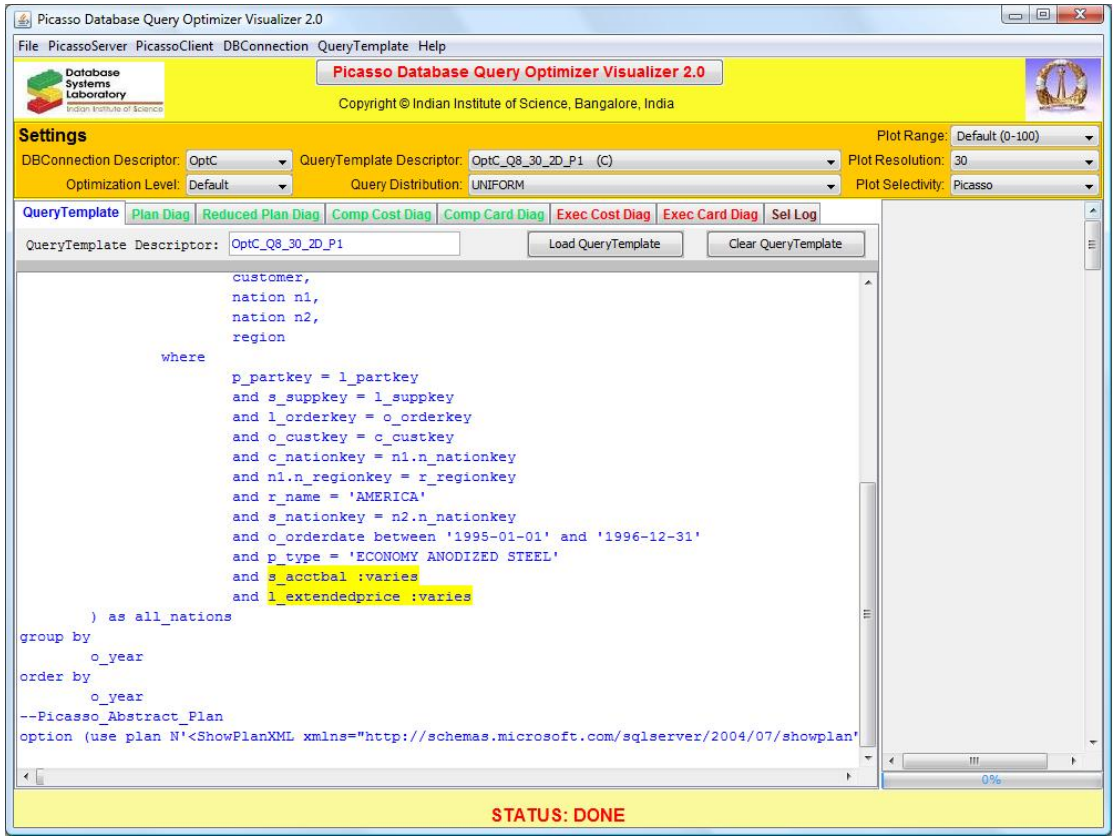
Ctrl + Right-click on a point in the diagram to view the ***Compiled Plan Tree***, that is, the detailed plan tree with costs and cardinalities for the individual nodes. This window also displays summary information about the overall estimated cost and result cardinality.

- ***Abstract Plan Diag:*** *Alt + Right-click* on a point to produce a new set of compilation or execution diagrams, where the plan associated with the clicked point is supplied to the engine as an ***Abstract Plan***, along with the associated query, for each query point in the new diagrams. This abstract plan is then used as a hint by the optimizer in plan selection, and is typically chosen in all (or most) of the entire space covered by the new diagrams. The utility of this feature is that the associated cost diagrams provide a visualization of the behavior of the cost function of the specific abstract plan over the entire selectivity space, and not just the regions in which it happens to be the optimal choice. When Abstract-Plan is invoked, Picasso automatically creates a new QTD by adding the suffix *"_Px",* where *x* is the plan number, to the original plan's QTD – this can be changed by the user if desired.

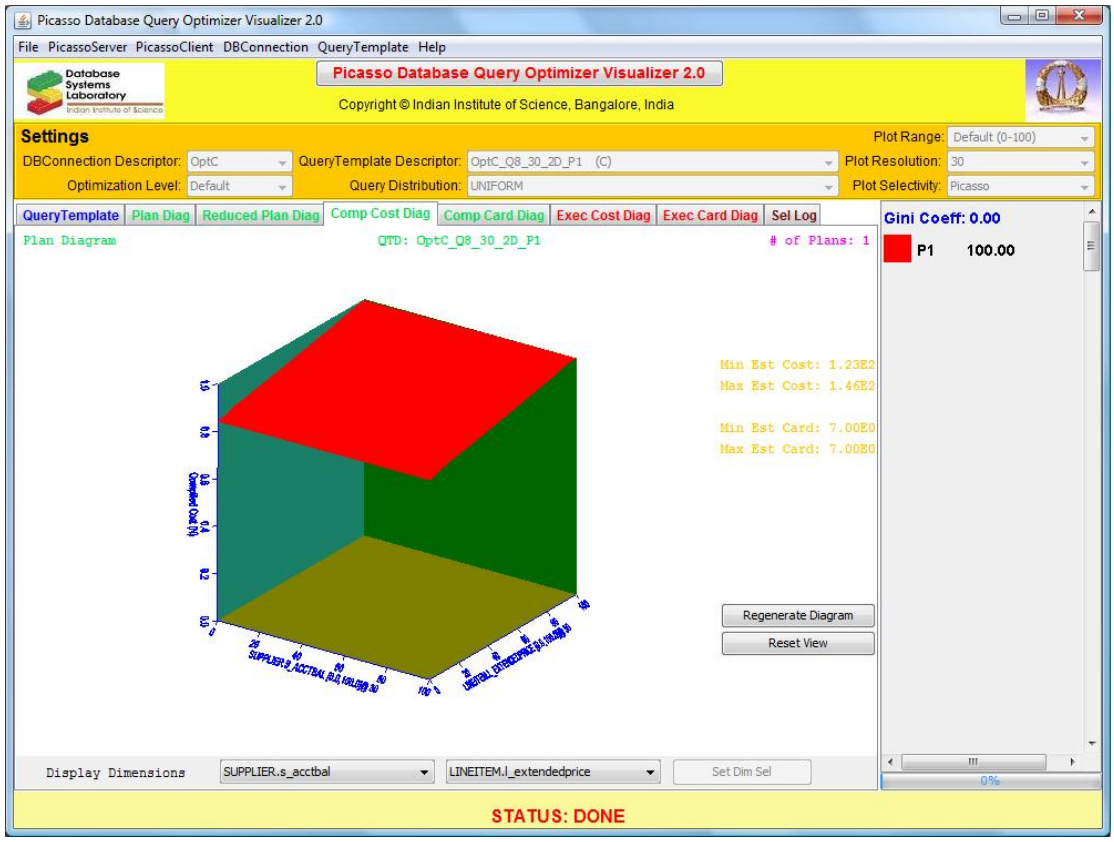
  Note: ***This feature is currently available only on SQL Server and Sybase ASE, which natively support the Abstract-Plan functionality at the engine API. A related point is that in SQL Server, the abstract plan may include the Picasso predicates with constants***

corresponding to the clicked point – this means that if the diagrams of the abstract plan are re-invoked from a different point In the original plan diagram, Picasso will treat it as a fresh diagram generation. In Sybase ASE, however, the abstract plan does not include such constants and hence the previously-produced diagrams of an abstract plan can be directly re-used in subsequent invocations. The details of Abstract-Plan functionality for SQL Server are available here, and for Sybase ASE they are available here .

An example query template incorporating an Abstract Plan is shown (see the last line of the template) in the picture below:



and an example cost diagram produced from such a template is shown in the following picture:



- ***Foreign Plan-tree Diagram:*** *Ctrl + Alt + Right-click* on a point in the diagram to compare the associated plan with the plan choice made at the same selectivity coordinates by another database engine, or by the same engine at a different optimization level (we refer to this new plan as ***Foreign Plan***). An engine and optimization level dialog first appears, and after the desired choices are selected, two plan trees are shown – the foreign plan and, to aid in comparison, the original local plan.
  With the dialog, the query template also appears and can be *syntactically edited* – the reason this editing may be required is that SQL dialects *vary across database engines*. For example, the SQL standard string function ***SUBSTRING*** is invoked as ***SUBSTR*** in some engines – a detailed listing of such dialect variations is available in this Wikibook .

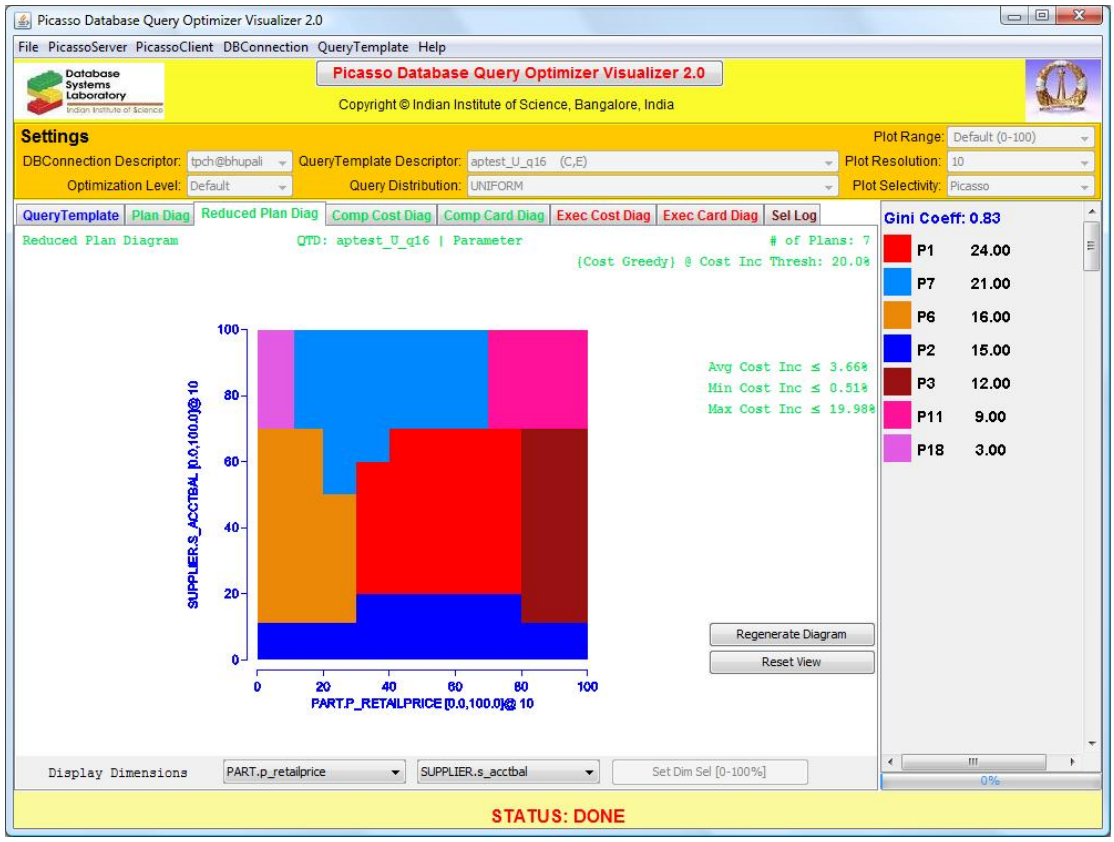
  While the editing can be done manually, a third-party automatic multi-dialect SQL conversion tool, SwisSQL API (Java), which supports all major database engines, can also be interfaced with Picasso. The SwisSQL tool can be obtained through email enquiry. It is available for payment under a commercial license, and free of charge under an academic license (a trial

version is also available for free download). To use SwisSQL with Picasso, download the software, store the ***SwisSQLAPI.jar*** library file in the Libraries folder, and edit the ***runClient.bat | runClient.sh*** files in the PicassoRun\Windows | PicassoRun/Unix directories to use the AutoConvert-inclusive run-time invocation. This will activate the ***AutoConvert SQL Dialect*** button on the dialog and the specific conversion carried out will be based on the user's choice of engine. Finally, hitting the ***Submit*** button invokes the foreign plan production process. [Note: As a general precaution, we recommend that users check that the auto-conversion result is satisfactory before submission; further, if the auto-conversion fails for any reason, manual conversion can always be used instead.]

Click the ***Reset View*** button to restore the original view of the diagram.

(Note: For Sybase ASE, the plans produced are compliant with those produced by the iSQL command-line client.)

- ***Reduced Plan Diag:***



The *Reduced Plan Diagram* shows the extent to which the original plan diagram can be simplified by completely replacing some of the existing plan regions with their sibling optimal plans in the diagram, *without increasing the cost* of any individual query by more than a user-specified threshold value.
When this tab is clicked, the ***Enter Cost Increase Threshold*** dialog box asks for the threshold value which can be any positive number (the default value is determined by the PLAN_REDUCTION_THRESHOLD macro in PicassoConstants.java – it is set to 10% in the distribution). For Cost-Bounding-based reduction algorithms (see below), a rough estimate of the location of the "knee" in the graph characterizing "Plan Cardinality of the Reduced Plan Diagram versus Cost Increase Threshold" is also provided as an aid to help the user in choosing the threshold. Further, an estimate of the threshold location to result in a desired number of plans is also given – this number is specified by the DESIRED_NUM_PLANS macro in PicassoConstants.java, which is set to 10 in the distribution.

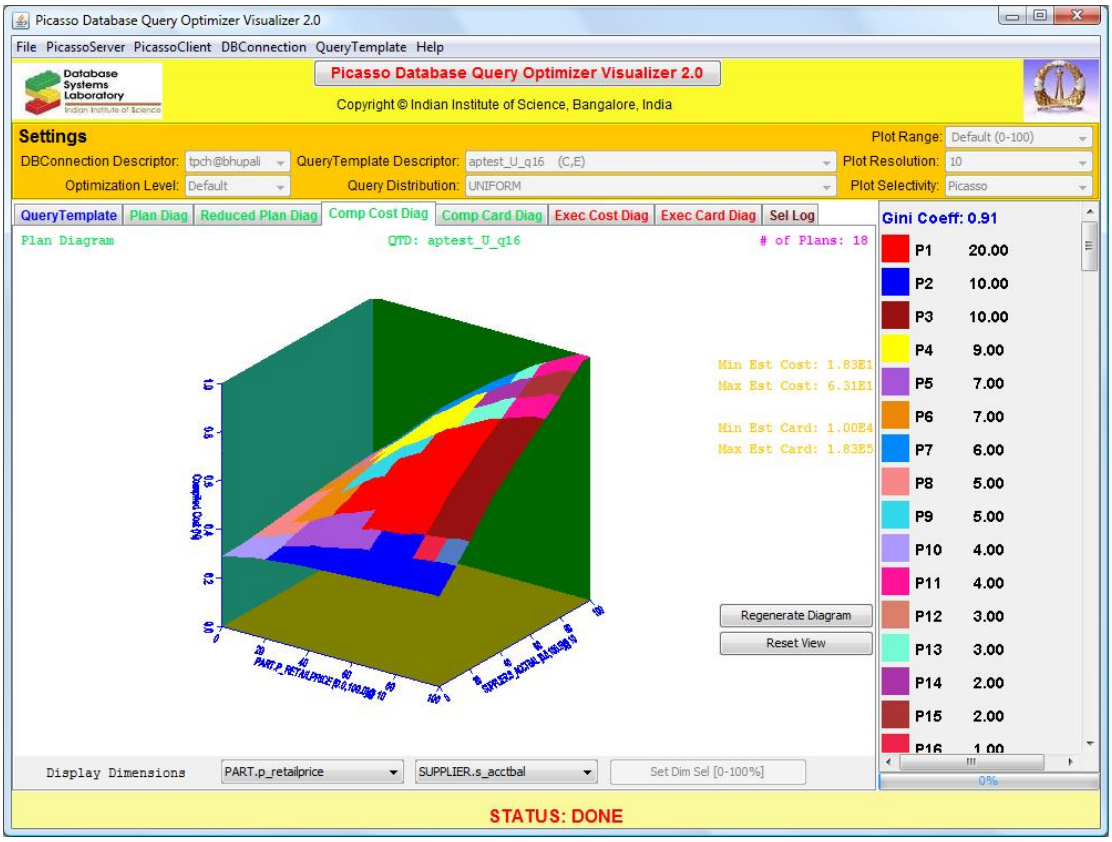
The Reduced Plan Diagram is labeled and graduated similar to the Plan Diagram. The QTD is displayed at the middle of the top panel, and the reduced number (#) of plans is displayed at the right corner. The bounds on the minimum, maximum and average cost increases are displayed in the middle of the right panel for Cost-Bounding-based reduction algorithms (see below).

The mouse-key-control operations are the same as those of the Plan Diagram, except that *"Compiled Plan Tree"*, *"Abstract Plan Diagram"* and *"Foreign Plan Tree"* cease to be valid choices.

A variety of choices of Plan Reduction algorithms are available in Picasso. They are: CostGreedy [VLDB 2007], CC-SEER and LiteSEER, the latter two derived from SEER [VLDB2008]. In the CostGreedy algorithm, ***bounds*** on the increased costs of swallowed points are ***inferred*** from the costs of points present in their neighborhood. On the other hand, the CC-SEER and LiteSEER algorithms explicitly evaluate the increased costs of swallowed points through the Abstract-plan-costing mechanism – therefore, they can be used only on engines that support this feature (SQL Server and Sybase ASE in Picasso). The required algorithm can be selected via Picasso Client → Local Settings. While the CostGreedy algorithm guarantees that the replacement plan is within the cost -increase threshold over the optimality region of the replaced plan, the CC-SEER algorithm provides the same guarantee over the ***entire selectivity space***, thereby providing robustness to errors in selectivity estimates, a common problem faced by query optimizers in practice. The LiteSEER algorithm is a heuristic variant of CC-SEER that substantially reduces the computational effort and provides robustness similar to SEER, albeit without the guarantee. (Note: In Picasso 1.0, a cost-bounding-based reduction algorithm called AreaGreedy [VLDB 2005] was also supported, but it has been discontinued since CostGreedy is superior in all performance aspects.)

For a description of the plan reduction algorithms, see Algorithmic Details, and for a discussion of plan reduction semantics, see Diagram Semantics.

- ***Comp Cost Diag:***



The *Compilation Cost Diagram* is a visualization of the associated estimated plan execution costs. This is similar to a Plan Diagram, but shows the estimated cost of executing the query-template over the selectivity space, as an additional dimension. The costs are normalized to the range `[0,1]` with respect to the maximum cost occurring anywhere in the ***d***-dimensional selectivity space, i.e. global normalization.

The QTD is displayed at the centre-top of the diagram and the number (#) of plans in the corresponding 2-D plan diagram slice is displayed at the right-hand top corner. The global minimum and maximum estimated values of the cost and cardinality of the query template are displayed in the middle of the right-hand panel of the diagram.

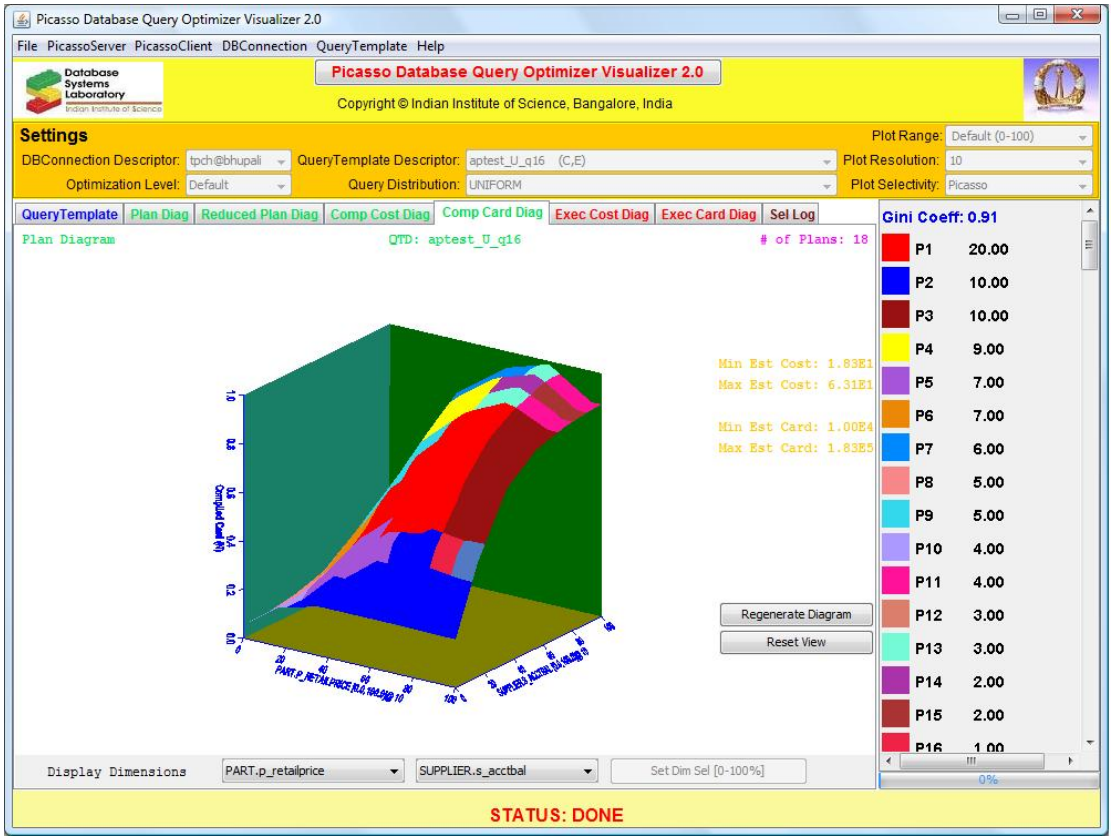
In this tab, you can:

> *Left-click* + *drag* on the diagram to rotate (or relocate the diagram in case of 1D query templates).
>
> *Shift* + *Left-click* on the diagram and move the cursor up to zoom in, or down to zoom out.
>
> *Ctrl* + *Left click* + *drag* to move it around.

Click the ***Reset View*** button to restore the original view of the diagram.

If the Cost Domination Principle (see the VLDB 2005 paper) is not followed in the diagram, an alert is issued to the user. However, to prevent unnecessary alarms arising out of arithmetic approximations, the threshold at which failure of cost domination is alerted is determined by the COST_DOMINATION_THRESHOLD macro in PicassoConstants.java – that is, the alert is issued only if the cost of a dominating point is less than (COST_DOMINATION_THRESHOLD * cost of any of its dominated points). The threshold is set to 95% in the distribution.
The alert also gives a count of the number of query points where cost domination was violated, and in the Client Console, gives the complete list of these violating points, indicating their costs and the costs of the points w.r.t. whom they have incurred the violation.

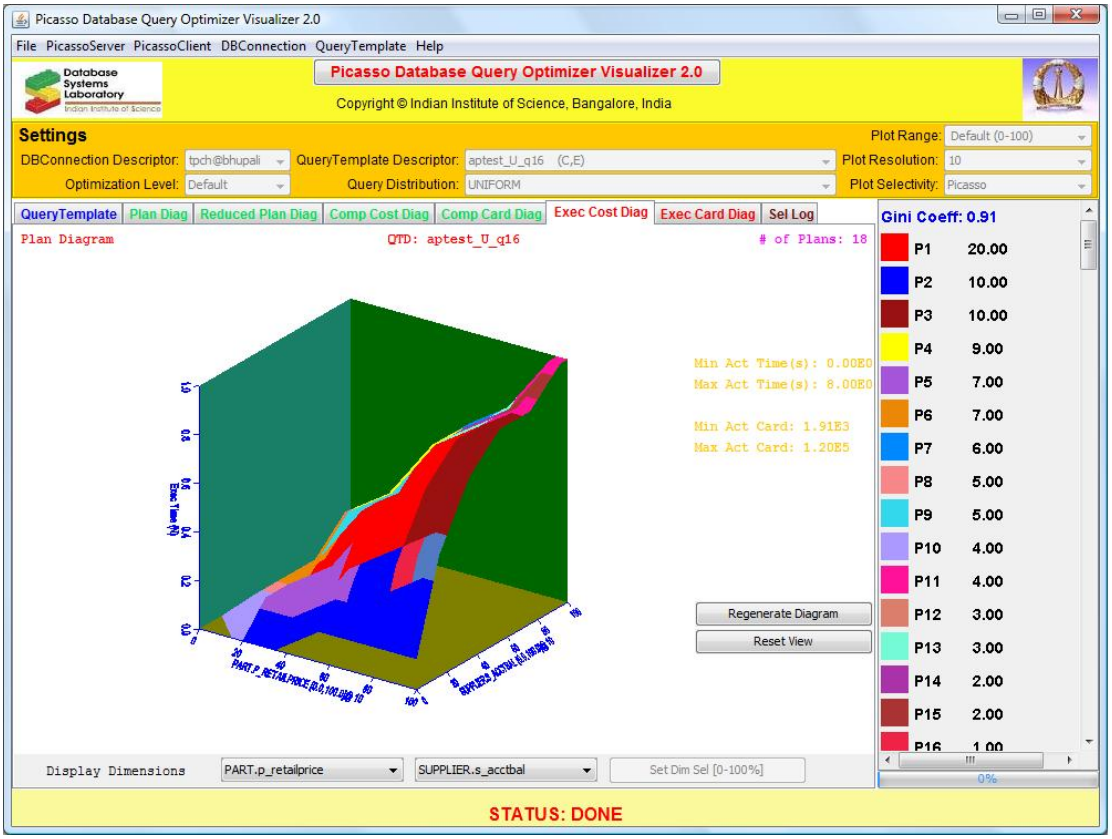
- ***Comp Card Diag:***



The *Compilation Cardinality Diagram* is a visualization of the associated

estimated query result cardinalities, and is similar in layout and behavior to a Compilation Cost Diagram.

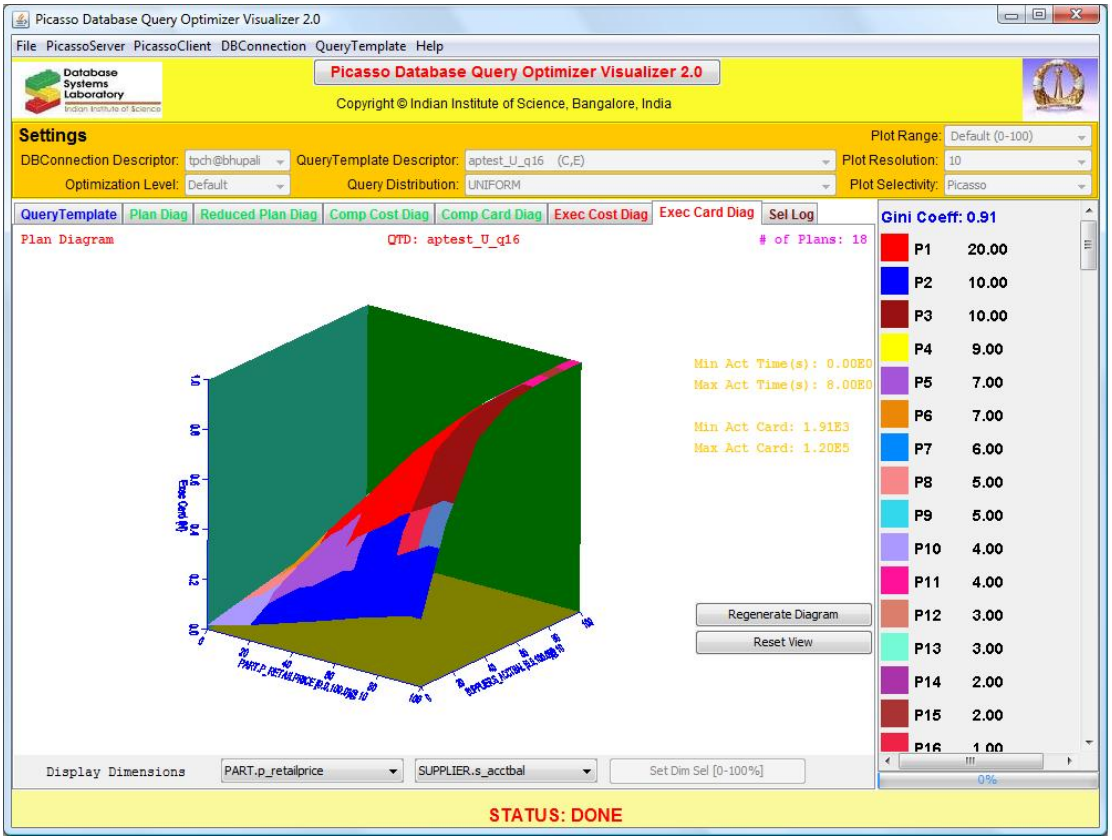
- ***Exec Cost Diag:***



The *Execution Cost Diagram* is a visualization of the query execution costs – evaluated in terms of response time – as determined through actual execution of the queries. The layout and behavior of this diagram is similar to that of the Compilation Cost Diagram. A comparison between the Execution Cost Diagram and the Compilation Cost Diagram can aid in characterizing the optimizer's modeling quality.

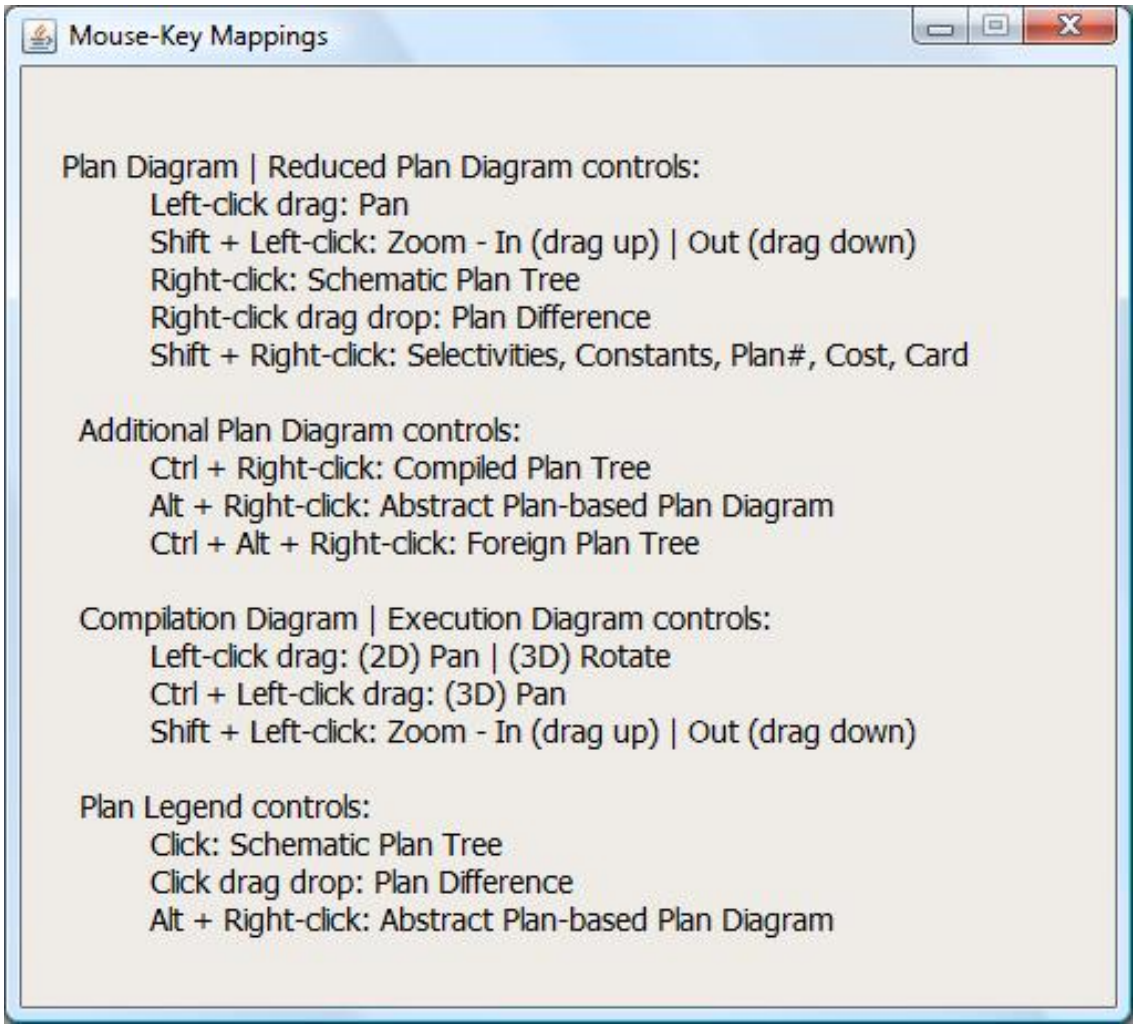
Notes:
1) The response time values in the Exec Cost Diag are given in units of seconds.
2) The running times can be highly dependent on the system status at the time of query execution and it is therefore possible that the execution cost diagrams for a given query template are *significantly different between successive generations*. For more details, see Diagram Semantics.

- ***Exec Card Diag:***



The *Execution Cardinality Diagram* is a visualization of the result cardinalities obtained through actual execution of the query, and is similar in layout and behavior to an Execution Cost Diagram. A comparison between the *Execution Card Diagram* and the *Compilation Card Diagram* can aid in characterizing the optimizer's modeling quality. Note that unlike the Execution Cost Diagram, the Execution Cardinality Diagram is not affected by the diagram production environment. In this sense, it provides a more stable characterization of the optimizer's modeling quality relative to the cost-based comparisons.

Summary of Mouse-Key controls for Picasso Diagrams

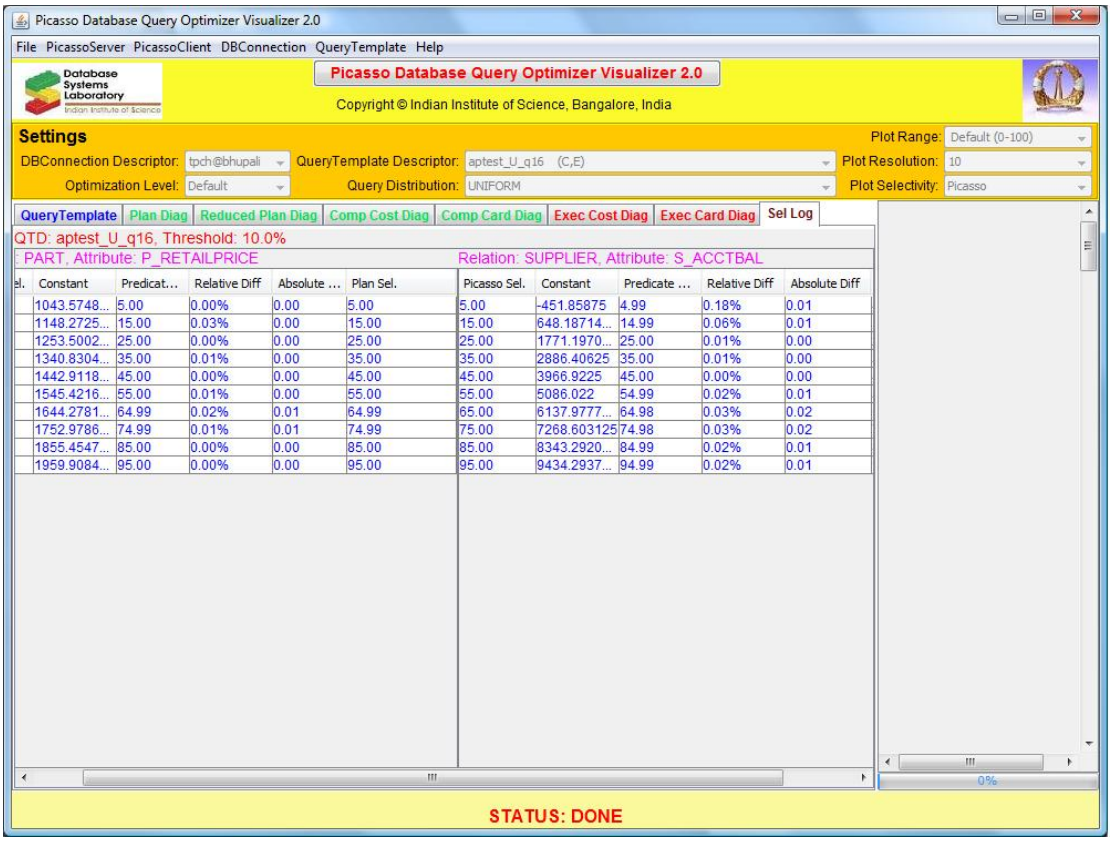


Notes:

1) If any of the diagram tabs are clicked when the associated pictures have not been previously generated, Picasso will estimate the time to generate the corresponding diagrams and ask for a confirmation of generation. If Yes is clicked, then the generation takes place, with the status bar showing the current status of completion, elapsed time and estimated remaining time, and the progress bar continuously showing the progress. But if No is clicked, the control is transferred to the QueryTemplate tab. See DiagramSemantics for a discussion of the estimation process.

2) The time estimates for producing an execution diagram are based on actually executing a sample query, and it may therefore take significant time to produce the estimate.

- ***Sel Log:***



When this tab is clicked, the Picasso selectivities and the associated constants are displayed for each PSP, along with the Predicate and Plan selectivities – see Diagram Semantics for details on these various selectivities.

The absolute and relative differences between the Picasso selectivities and the Predicate selectivities are shown as Absolute Difference and Relative Difference, respectively. Rows that have a Relative Difference above a user-specified difference threshold are marked in red. The relative difference threshold is specified using the SELECTIVITY_LOG_REL_THRESHOLD macro in PicassoConstants.java – it is set to 10% in the distribution.

Note: Whenever the Absolute Difference is less than the value of the SELECTIVITY_LOG_ABS_THRESHOLD macro in PicassoConstants.java (set to 1% in the distribution), the corresponding rows are *not shown in red* even if the relative difference is more than the threshold – this is to prevent unnecessary alarms arising from small selectivity values.

6. Remaining Buttons and Indicators

- ***Regenerate Diagram button:*** This is found in all the 6 diagram tabs, near the right-hand bottom side of the screen. Clicking this button in any of the plan or compilation diagrams regenerates the diagrams, reflecting changes (if any) in settings or environment. Clicking the button in any of the execution diagrams regenerates all the diagrams (both compilation and execution).

- ***Reset View button:*** This button lies below the ***Regenerate Diagram*** button in all the diagram tabs. Clicking it restores the original view of the diagram.

- ***Gini Coeff:*** This is the Gini coefficient, displayed just above the plan legend, which measures the ***skew*** in the areas of the regions covered by the various optimal plans in the plan and reduced plan diagrams. It is a number between 0 and 1, where 0 corresponds to no skew (i.e. all plans cover approximately the same area) and 1 corresponds to extreme skew (i. e. one plan covers almost the entire space).

- ***Plan Legend:*** Occupies a vertical strip near the right-hand side of the screen. This is the legend for the current Plan Diagram, but is displayed alongside all the other diagrams as a visual aid. (For the Reduced Plan Diagram, the corresponding legend with the reduced number of plans is shown.) Each colored square represents a plan, and the associated number and space coverage in the current diagram is shown alongside. The plans are displayed in decreasing order of space covered. Here too, as in the case of the Plan Diagram, you can: (a) click on a color to view the associated ***Plan Tree*** window; (b) click on one plan, drag and drop the mouse on a second plan to view the ***Plan Difference*** window showing both the plans, with the differences between the plans highlighted; and (c) *Alt +Right-click* to invoke the ***Abstract Plan*** feature.

- ***Progress Bar:*** This is found below the legend. When any diagram is being generated, the progress of that process is shown here with a growing progress bar and the percentage completed.

- ***Status Bar:*** This is the bottom-most horizontal strip in the screen. It shows the current status in red colored text on a yellow background.

- ***Cancel Processing Button:*** This button located at the bottom portion of the screen above the status bar, becomes active during the diagram generation or reduction process. Clicking it terminates the current process at both client and server and discards all information gathered thus far.

- ***Pause/Resume Processing Button:*** This button located at the bottom portion of the screen above the status bar, becomes active during the diagram generation or reduction

process. Clicking it makes the server pause the current process. The process can be resumed by clicking the button again.

7. Plan Tree windows

Click here for information on the various Plan Tree windows.

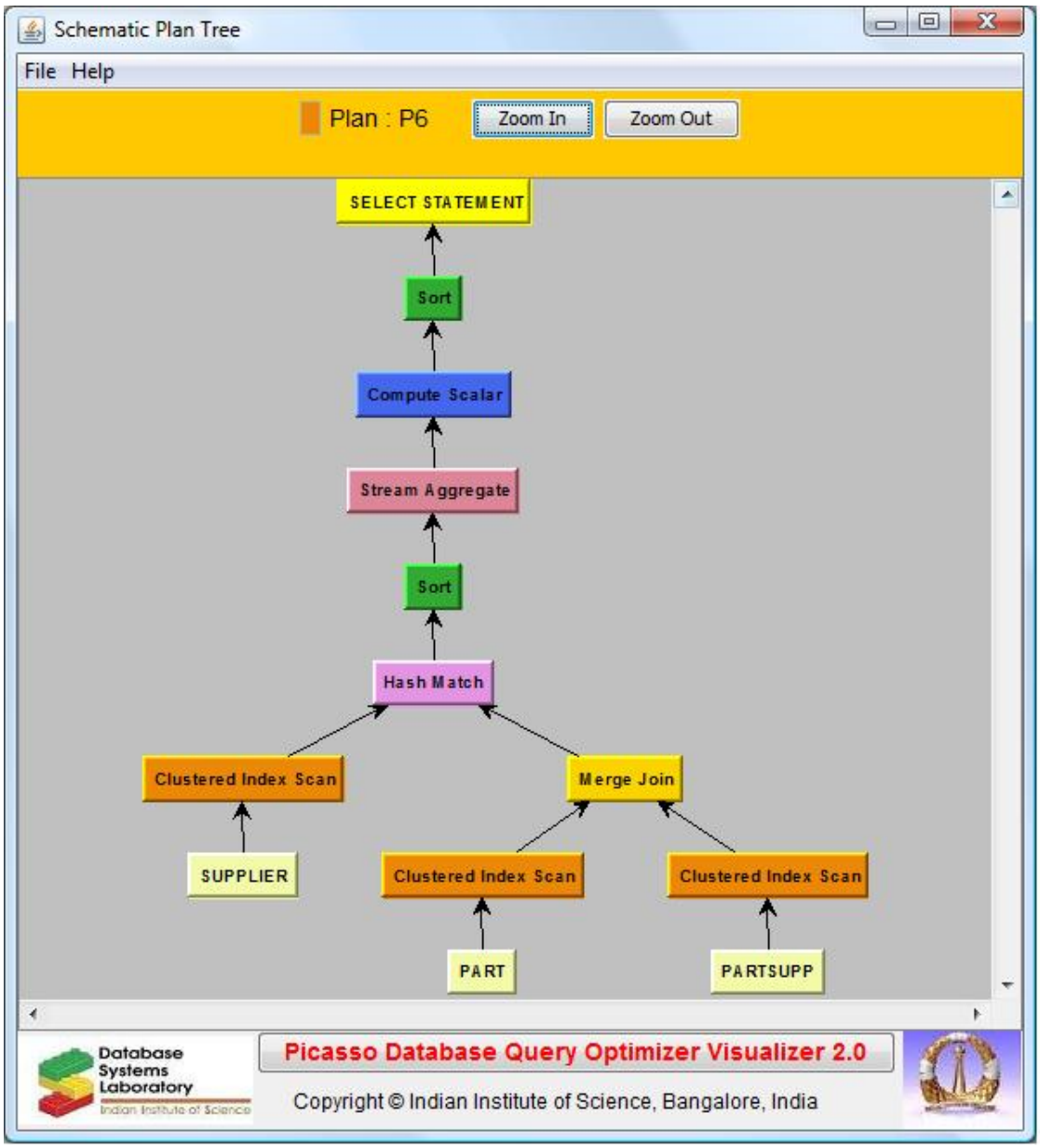
Documentation Home

Picasso Database Query Optimizer Visualizer

PLAN TREE WINDOWS

This section describes the various plan-related windows: Schematic Plan Tree, Compiled Plan Tree, Plan Difference, and Foreign Plan Tree that arise from keyboard-mouse operations on the Picasso diagrams.

1. Schematic Plan Tree window:



This window appears when any plan region in ***PlanDiag*** or ***Reduced PlanDiag*** is right-clicked and displays the schematic structure of the associated plan tree. The window also appears when any plan in the legends that are shown in all the compilation diagrams (PlanDiag, Reduced PlanDiag, CompCost, CompCard) is clicked. The window has these controls:

Zoom In button: Click to enlarge the tree.

Zoom Out button: Click to shrink the tree.

Node Information: When a node in the plan tree is selected, this panel shows the name of the node and its parameter details, if any.

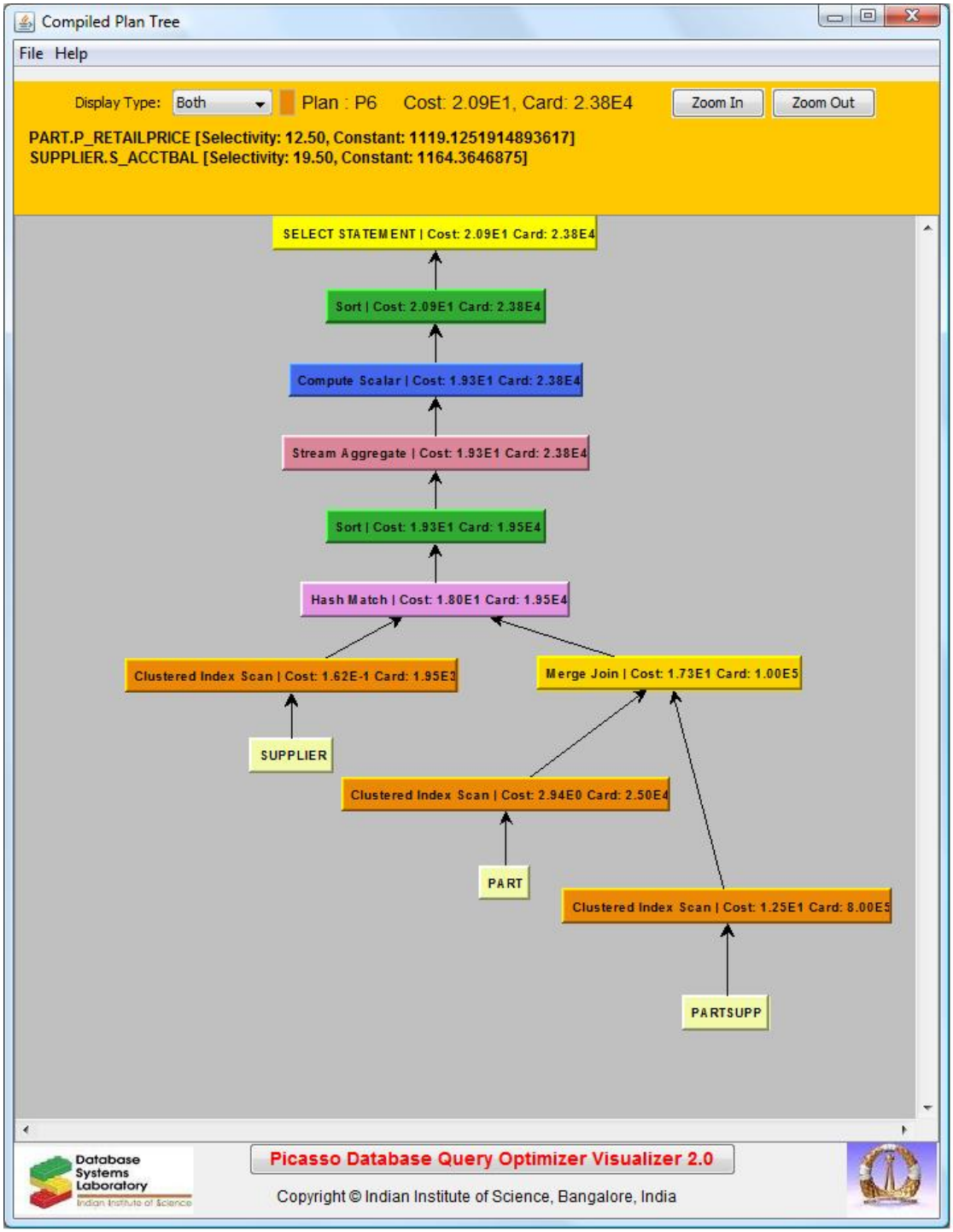
Plan Tree: This is the pictorial representation of the plan tree. Each node in this tree represents an execution operation and is displayed with its name. Further, each node type is assigned a unique color. Clicking on a node displays its details (including parameter information, if any) in the space above the tree. If necessary, the nodes and edges in the tree can be moved around to enhance the display clarity.

Note: The names of the nodes are retained the same as those used by the respective database engines. That is, similar operators may be named differently across plan trees corresponding to different database engines (e.g. DB2 uses RETURN and TBSCAN to refer to what Oracle chooses to call SELECT STATEMENT and TABLE ACCESS, respectively.) Refer the vendor documentation for the semantics of the various node types.

Menu bar:
File menu: There are 4 menu items: ***Save***, ***Print***, ***Print Preview*** and ***Exit***. Their functions are the same as those in the root client window.
Help menu:

Usage Guide: Opens a help page pertaining to Plan Windows.
About Picasso: Displays a dialog containing information on Picasso including version, home page, etc.

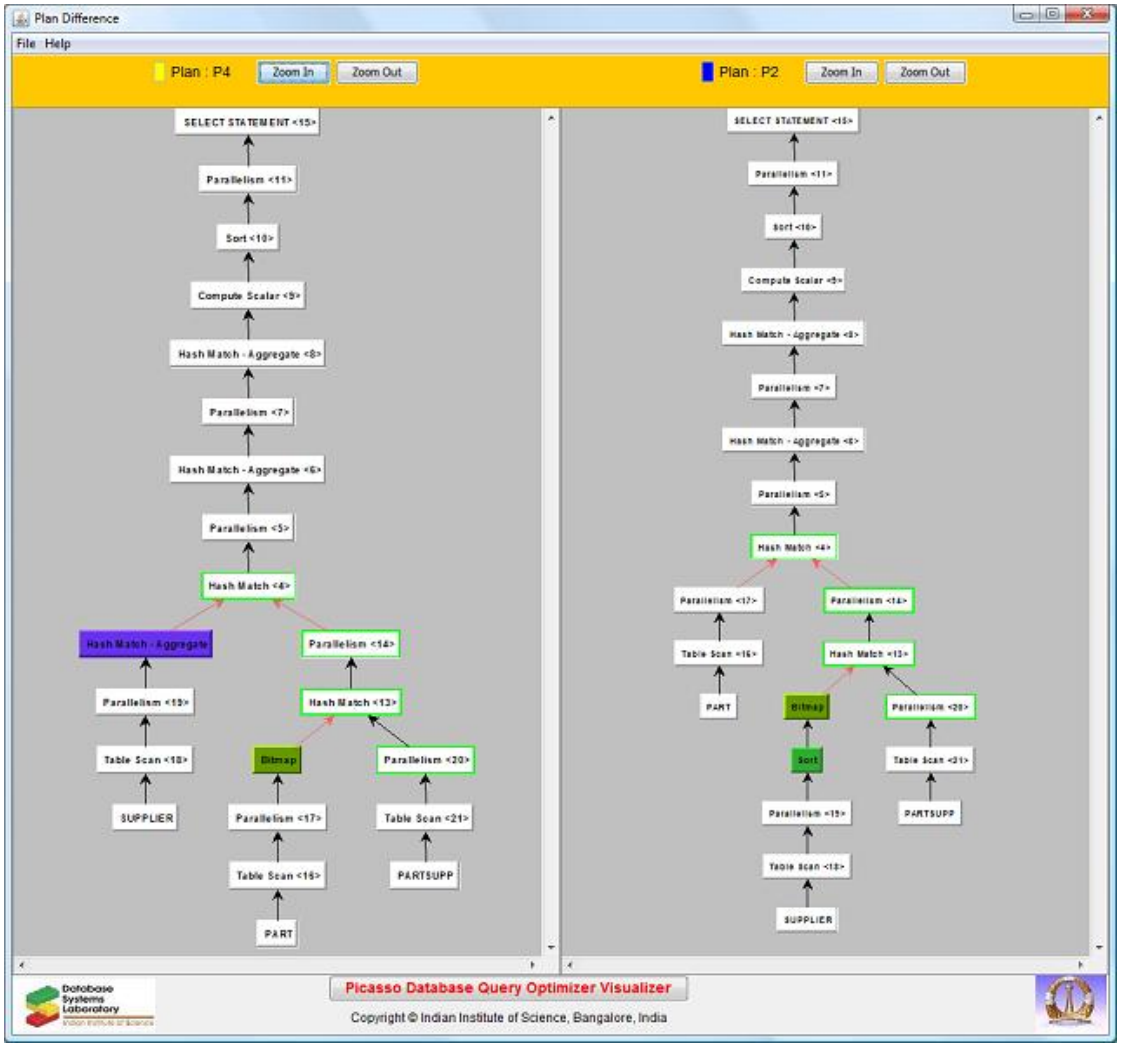
2. Compiled Plan Tree window:



This window appears when the user executes a *Shift+right-click* on any point within the ***PlanDiag***. This window has the same controls as the Schematic Plan Tree, with the addition of a ***Display Type*** drop-down box – the selection in this box determines what information is displayed for each node. The default setting is *'Both'*, whereby the cost and cardinality at each node is shown. With *'Cost'* only the cost is shown, while with *'Card'* only the cardinality is shown, and with *'None'*, the vanilla plan tree is displayed. The difference between Compiled

Plan Tree and Plan Tree is that the compiled plan tree is with respect to a specific query point, whereas the plan tree is with respect to a plan region.

3. Plan Difference window:



This window appears when the user *right-clicks* on a particular plan (either in the diagram for ***PlanDiag*** and ***Reduced PlanDiag***, or in the legend for all the compilation diagrams) and then drags the mouse and drops on a different plan. It shows side-by-side the schematic plan trees corresponding to these two plans. The nodes which perform the same function (i.e. same label) with the same inputs in both the plans are shown completely in *white*, signifying their commonality. The differences, on the other hand, are highlighted through different colors, as given below:

- Node labels, Node parameters and Node inputs are identical: White nodes with Black input links
- Node labels are the same: White Fill
    - Parameters different: Green-bordered nodes with Black input links
    - Left and right inputs swapped: Orange-bordered nodes with Blue input links
    - Left and right inputs different: Orange-bordered nodes with Red input links
- Node labels are different: Red-bordered nodes filled with the native node color
    - Left and right inputs swapped: Blue input links
    - Left and right inputs different: Red input links
- Nodes are unmapped (i.e. no corresponding node in the other tree): Nodes filled with the native node color and Black input links

The controls in this window are the same as those in the Schematic Plan Tree window, with an additional item, ***Color Code Guide,*** in the ***Help*** menu, which displays the above color coding.

Note that when plan difference is carried out on a ***OperatorDiff***-based plan diagram, the trees will always include at least one colored node or link (that is, unlike ***ParameterDiff***-based plan diagrams, it is not possible to have pure black-and-white trees with only green node borders).

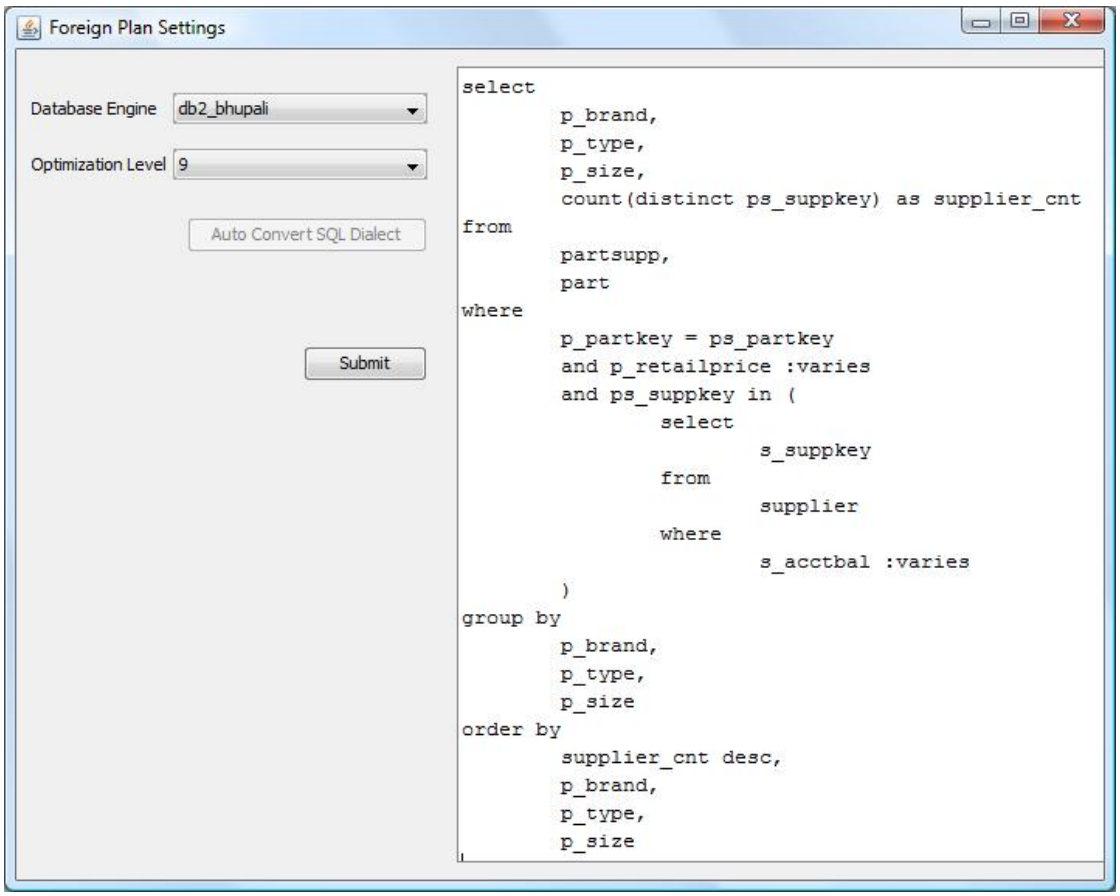
For a discussion of Plan Difference semantics, please see Diagram Semantics, and for a description of the node-matching algorithm, see Algorithmic Details.

4. Foreign Plan Tree window:

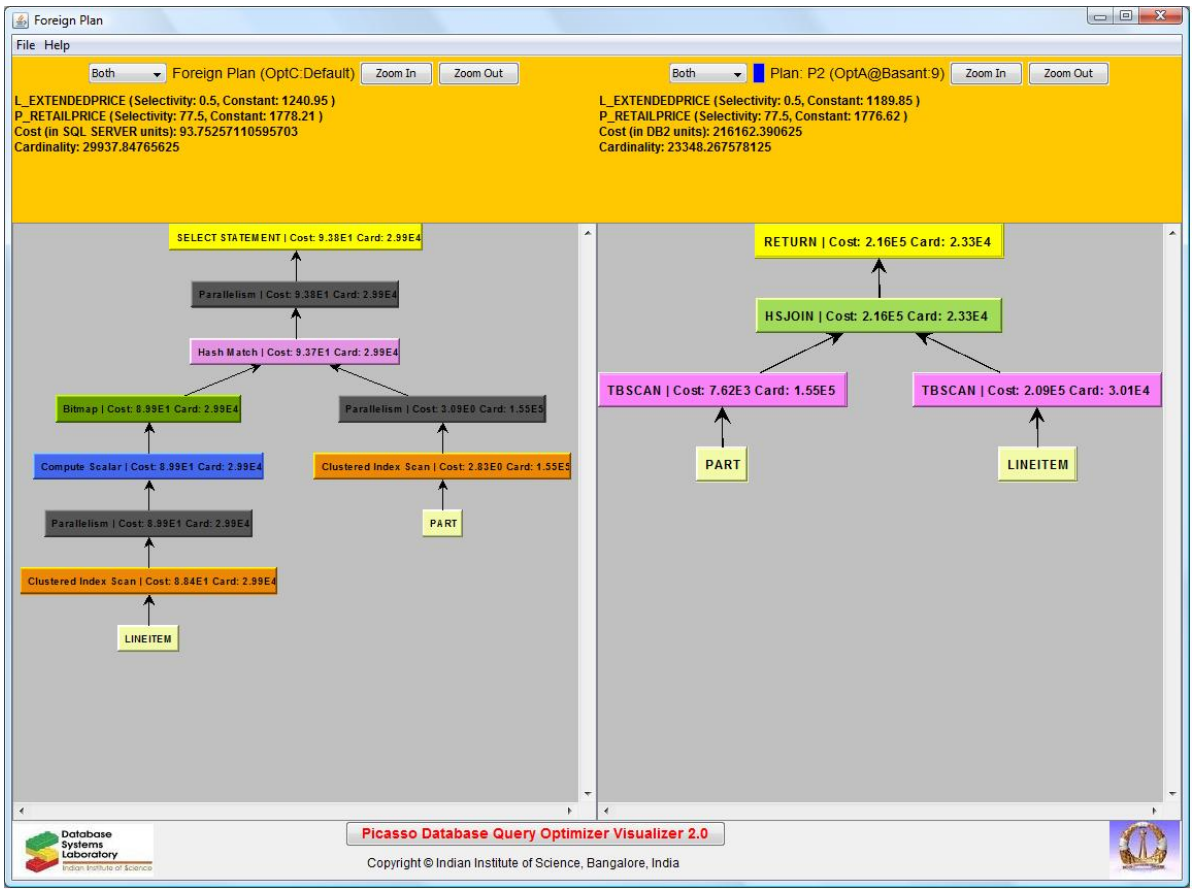
This window appears when the user executes a *Ctrl+Alt+right-click* on any point within the ***PlanDiag*** and chooses either a foreign engine, or the same engine at a different optimization

level. The window shows the compiled versions of both the foreign plan-tree and the local plan-tree at the selectivity coordinates of the clicked point. If a common engine is used, then the schematic plan difference between the two trees is also shown.

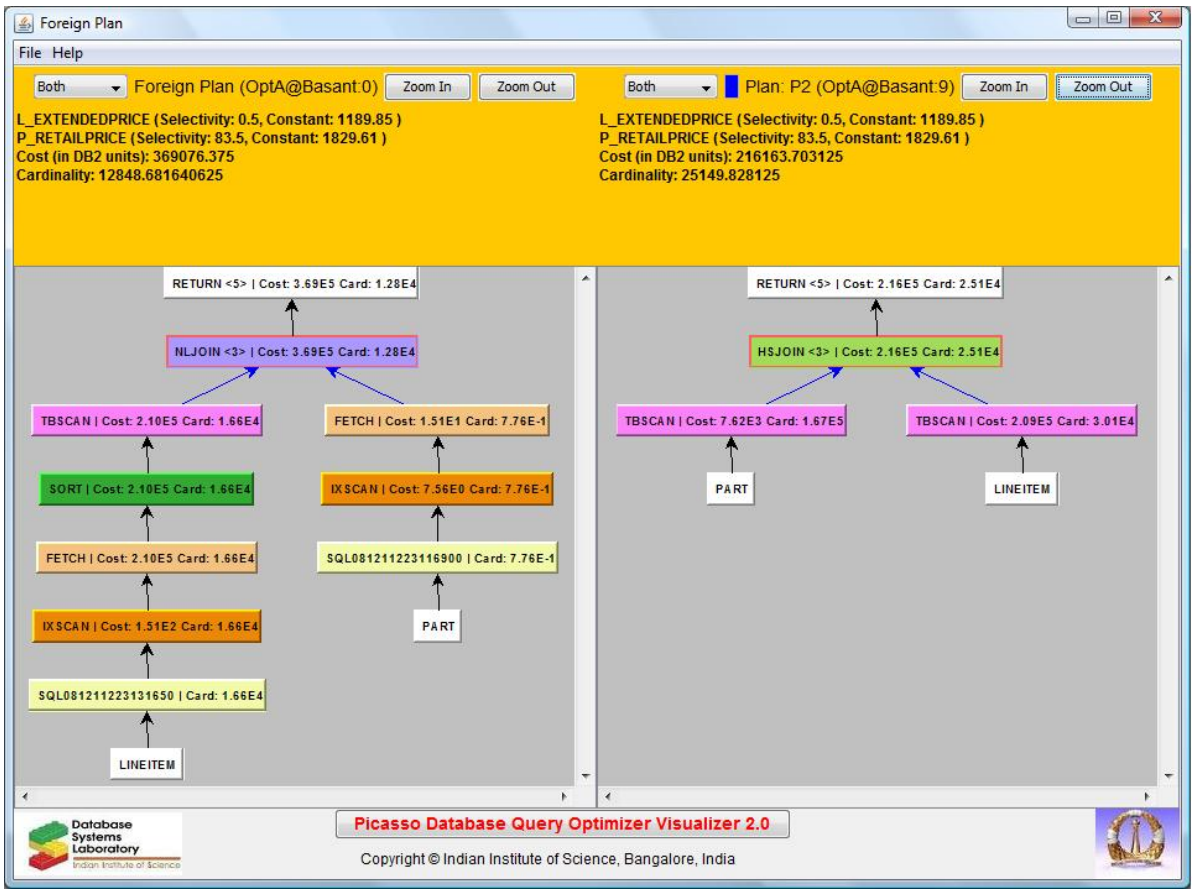
The settings panel for this functionality is shown below.



- Example Foreign Plan Tree Diagram for the Different Engines case:



- Example Foreign Plan Tree Diagram for the Same Engine case (note the Plan Difference comparison):



Documentation Home

Picasso Database Query Optimizer Visualizer

COMMAND LINE INTERFACE to PICASSO SERVER

Apart from the Picasso Client graphical interface described in the main documentation, the Picasso Server can also be accessed directly through the command line. The utility resides in the PicassoRun directory and is called **PicassoCmd.**

The command line utility can be used as follows:

1. a) For generating exact diagrams that have the same resolution and the complete range on all dimensions:
   PicassoCmd <ServerName> <Port> <DBConnection> <OptLevel> <QTD> <QDist> <DiagType> <QTFile> <Resolution>

   b) For generating approximate diagrams that have the same resolution and the complete range on all dimensions:
   PicassoCmd <ServerName> <Port> <DBConnection> <OptLevel> <QTD> <QDist> <DiagType> <QTFile> <Approx-algo> <IdError> <LocError> <Resolution>

2. a) For generating exact diagrams with dimension-specific ranges and/or resolutions:
   PicassoCmd -R <ServerName> <Port> <DBConnection> <OptLevel> <QTD> <QDist> <DiagType> <QTFile> <Approx-algo> <IdError> <LocError> {<Resolution> <Startpoint> <Endpoint>}

   b) For generating approximate diagrams with dimension-specific ranges and/or resolutions:
   PicassoCmd -R <ServerName> <Port> <DBConnection> <OptLevel> <QTD> <QDist> <DiagType> <QTFile> {<Resolution> <Startpoint> <Endpoint>}

The number of {<Resolution> <Startpoint> <Endpoint>} triplets has to be identical to the dimensionality of the query template. The triplets are associated with the PSP predicates in their *syntactic* order of appearance in the query template – i.e., the first triplet corresponds to the first :varies predicate in textual order, the second triplet corresponds to the second :varies predicate, and so on.

The following table explains the various arguments:

Argument	Meaning
ServerName	Name/IP of machine running Picasso Server
Port	Port number of Picasso Server
DBConnection	Database Connection Descriptor
OptLevel	Database Engine's Optimization Level
QTD	Query Template Descriptor
QDist	Query Distribution (permitted values are Uniform and Exponential)
DiagType	Diagram Type (permitted values are Compilation, Approximate and Execution)
QTFile	File containing the Query Template
Approx-algo	Choice of Approximation algorithm (permitted values are Sampling and Grid)
IdError	Identity error tolerance (1-99 percent)
LocError	Location error tolerance (1-99 percent)
Resolution	Number of query points along a dimension (permitted values are 10, 30, 100, 300 and 1000)
Startpoint	Start point of the selectivity space along the dimension (0-99 percent)
Endpoint	End point of the selectivity space along the dimension (1-100 percent). Endpoint must be greater than Startpoint by at least 1%.

Notes:

1. The first eight arguments and <Resolution> are compulsory for all diagrams. Further, if DiagType = ***Approximate***, then the arguments <Approx-algo>, <IdError>, <LocError> are compulsory.
2. If the -R option is used, then the number of {<Resolution> <startPoint> <endPoint>} triplets must be exactly the same as the dimensionality of the query template.

The following are representative examples of using the command line facility:

a. PicassoCmd localhost 4444 sql_localhost default SQL_Default_Uniform_100_q2 Uniform Compilation E:\Picasso \QueryTemplates\sqlserver\q2.sql 100

first connects to the Picasso server residing on "localhost" at port "4444" and then to the database engine using the "sql_localhost" connection descriptor. It then produces compilation diagrams with QTD "SQL_Default_Uniform_100_q2" at a "default" optimization level with queries Uniformly distributed at a resolution of 100 on each dimension over the entire selectivity range, for the query template in file "E:\Picasso\QueryTemplates\sqlserver\q2.

sql".

b. PicassoCmd localhost 4444 sql_localhost default SQL_Default_Uniform_100_q8_RS-NN Uniform Approximate E:\Picasso\QueryTemplates\sqlserver\q8.sql Sampling 10 20 100
first connects to the Picasso server residing on "localhost" at port "4444" and then to the database engine using the "sql_localhost" connection descriptor. It then produces approximate diagrams with QTD "SQL_Default_Uniform_100_q8_RS-NN" at a "default" optimization level with queries Uniformly distributed at a resolution of 100 on each dimension over the entire selectivity range, for the query template in file "E:\Picasso\QueryTemplates\sqlserver\q2.sql" using the "Sampling (RS-NN)" approximation algorithm with tolerances of 10% identity error and 20% location error.

c. PicassoCmd -R localhost 4444 sql_localhost default SQL_Default_Uniform_100_q9_custom Uniform Compilation E:\Picasso\QueryTemplates\sqlserver\q8.sql 100 10 80 30 20 50
first connects to the Picasso server residing on "localhost" at port "4444" and then to the database engine using the "sql_localhost" connection descriptor. It then produces compilation diagrams with QTD "SQL_Default_Uniform_100_q9_custom" at a "default" optimization level with queries Uniformly distributed for the query template in file "E:\Picasso\QueryTemplates\sqlserver\q9.sql". The diagram has customized range and resolution on different dimensions – in the first dimension, the resolution is 100 with the selectivity range going from 10% to 80%, while in the second dimension the resolution is 30 with the selectivity range going from 20% to 50%.

Documentation Home

Picasso Database Query Optimizer Visualizer

DIAGRAM SEMANTICS

This section describes the semantics underlying the various Picasso diagrams.

1. Diagram Production

Given a ***d***-dimensional query template and a plot resolution of n_i on dimension ***i***, the Picasso Server generates $\prod_{i=1}^{d} n_i$ queries that are either uniformly or exponentially (based on the user's choice of ***Query Distribution***) distributed over the selected selectivity space. In the case of compilation diagrams, each of these queries is submitted to the database engine to be optimized, whereas in the case of execution diagrams, the queries are actually executed. After the plans corresponding to all the points are obtained, a different color is associated with each unique plan, and all query points are colored with their associated plan colors. Then, the rest of the diagram is colored by painting the hyper-rectangle around each point with the color corresponding to its plan. For example, in a 2-D plan diagram with a uniform grid resolution of 10, there are 100 real query points, and around each such point a square of dimension 10x10 is painted with the point's associated plan color. The implementation in Picasso is such that the query instances are generated in *row-major order* beginning with the point closest to the origin.

Note that the above description is true for producing exact diagrams, the only option in Picasso v1. However, in version 2, a new feature of approximate compilation diagrams is supported. In these approximate diagrams, only a subset of the points in the selectivity space is explicitly optimized while the characteristics of the remaining points are *inferred.* The quality of the approximation and the associated overheads are determined by the error tolerances

specified by the user.

For each database on a database engine, all Picasso-related information is stored in a set of tables and views within the database on the engine – the tables are **PicassoQTIDMap, PicassoPlanStore, PicassoPlanTree, PicassoPlanTreeArgs, PicassoSelectivityMap, PicassoSelectivityLog, PicassoRangeResMap, PicassoXMLPlan, PicassoApproxMap** and the view is **picasso_columns**.

2. Picasso/Predicate/Plan Selectivities

Given a query point in the selectivity space, Picasso estimates the constants that would result in the desired selectivities of the base relations by essentially carrying out an "inverse-transform" of the statistical summaries corresponding to the Picasso columns (PSPs). However, it is possible that there are discrepancies between the optimizer's view of the selectivities corresponding to these constants and the Picasso view for a variety of reasons:

1. Different interpolation mechanisms from the summaries – Picasso always uses *linear* interpolation (for the commercial optimizers, such internal information is not readily available).

2. Different usage of the summaries – for example, some optimizers choose to not use the histogram statistics at low optimization levels, whereas Picasso always uses the complete information.

3. Values presented in the optimizer's query plan may sometimes be a function of the overall plan structure – for example, when an engine chooses a nested mode of execution with a PSP in the inner sub-query, the selectivities shown may not correspond to the range of the PSP, but to the equality predicate that is checked in each iteration of the nested execution.

4. Incorrect calculations in the optimizer – for example, one of the optimizers makes a miscalculation whenever the constant happens to coincide exactly with

histogram bucket boundaries, resulting in the clearly impossible situation where even with *C1 < C2,* the selectivity of *R.A ≤ C2* is estimated to be less than the selectivity of *R.A ≤ C1*, because *C2* happens to coincide exactly with the bucket boundary.

5. Sometimes, for some of the selectivity locations in a given plan diagram, the Plan Selectivities (see definition below) may be quite different from the corresponding Picasso or Predicate selectivities (see definition below) because the plans at those selectivity points may impose predicates in addition to the PSP on the node. This can happen, for example, when the PSP features in the inner relation of a nested loop join. Another reason for differences is that the plan selectivities are computed on a ***d***-dimensional space, whereas the Picasso and Predicate selectivities are computed on uni-dimensional space, and therefore the query points on which the plan selectivities are computed may produce plans that have issues like the nested-loop join situation. Finally, it is also possible that the optimizers may, for certain plan structures, produce rule-based selectivities.

To provide users with a holistic view of this issue, Picasso gives three selectivities: ***Picasso***, ***Predicate*** and ***Plan***, for the set of constants corresponding to each PSP, in the Selectivity Log that is produced along with the diagrams. These selectivities are defined as follows:

- Picasso selectivity: This is the selectivity of the PSP as determined by the Picasso software through the statistical summaries of the database engine.

- Predicate selectivity: This is the optimizer's estimated selectivity for the PSP when the uni-dimensional query "***select * from Table T where PSP(T)***" is optimized.

- Plan selectivity: This is the selectivity associated with the *node* containing the PSP in the optimizer's plan for the original query. The plans are taken from the query points on the *diagonal* of the ***d***-dimensional space.

3. Plan Difference

The semantics for 'plan difference' are *vendor-specific* in terms of what variations qualify as genuine semantic differences between a pair of plans. In the absence of such information, Picasso currently supports two extreme levels of differentiating between plans: As a "lower bound", it compares the plans at an Operator level, where only differences between either the tree structures themselves, or the node identities at corresponding locations, are taken into account. On the other hand, as an "upper bound", it compares the plans at a Parameter level, where auxiliary information associated with a node is also taken into account. To make this clear, consider a ***TableScan*** node in plan P_1 that has its ***Prefetch*** attribute set to ***'None'***, while in plan P_2 the same node has ***Prefetch*** set to ***'Sequential'***.

These two nodes would be identical with an Operator perspective, but different from a Parameter perspective.

A related issue in the above is "what constitutes an operator parameter", that is, what auxiliary information from the plan is to be associated with a node. Again, since this is vendor-specific, we have made choices based on our understanding of the relevance of the information to the execution engine. However, we recommend users of Picasso to suitably modify the code if they prefer alternative choices. Currently, the operator parameters included for the various engines are the following:

- DB2 – all operator parameters in the **EXPLAIN_ARGUMENTS** table except NUMROWS, BITFLTR, FETCHMAX, ISCANMAX, MAXPAGES, MAXRIDS, SPILLED (if a parameter is present multiple times for a given operator, only the first is retained)

- Oracle – all operator parameters present in the **OPTIONS** table

- SQL Server – all information associated with the operator (the constants associated with PSP predicates are replaced by *:varies*)

- Sybase ASE – the CacheStrategy and Order operator parameters

- PostgreSQL – no operator parameters are included since we were unable to identify any parameter information in the optimizer's plan output; as a consequence, there are no differences in the pictures derived with Operator and Parameter settings for this engine

4. Plan Reduction

In plan reduction, a plan in the plan diagram is replaced if and only if ***all*** its associated query points can be 'swallowed' by one or more sibling plans, without increasing the original cost of the points by more than the user-specified threshold percentage. The query points of a swallowed plan are re-colored with the colors of the "swallower" plans.

There are two categories of plan reduction algorithms:

1. Algorithms based on Cost-bounding (CB), wherein bounds on the increased costs of swallowed points are inferred from the costs of points present in their neighborhood. The AreaGreedy and CostGreedy algorithms implemented in Picasso fall into this category.

2. Algorithms based on Abstract-plan-costing (APC), wherein the increased costs of swallowed points can be explicitly evaluated through the Abstract-plan-costing mechanism. The CC-SEER and LiteSEER algorithms implemented in Picasso fall into this category.

Cost-bounding Approach

In the CB-based approach, the swallowing criteria used for a query point q_s is that the candidate swallower plan must have at least one query point q_t in the ***first quadrant relative to q as the origin in the selectivity space*** whose cost is within the threshold w.r.t. q's cost. If there are multiple such points, the q_t point with the lowest cost is selected as the replacement. Further, if there are multiple feasible swallower plans, the plan with the lowest replacement cost is chosen as the

preferred swallower.

A fundamental assumption in this reduction approach, which is also the reason for restricting attention to the first quadrant, is that *query processing costs monotonically increase with increasing selectivities* – that is, more input from the base relations implies more work to be done and hence more cost incurred (see the Cost Domination Principle in this VLDB 2005 paper). While this is true in practice for most query templates, it may sometimes not hold because of either (a) the intrinsic nature of the query template (e.g. presence of the EXISTS operator may lead to cost *reductions* with increasing selectivities), or (b) due to erroneous modeling in the optimizer. Further, even if it *apparently holds* based on the explicit query points that are optimized in the plan diagram, it is possible, albeit unlikely, that there are regions *in between* these points where it may be violated. Therefore, the reduction diagrams should always be *interpreted with care* keeping in mind the likelihood of the cost monotonicity assumption being valid. Note that explicit violations of the assumption are flagged by the Cost Domination Principle violation alert message in Picasso. Further, in such situations where the costs do not monotonically increase in the entire space, we permit swallowing only if the candidate swallower point in the first quadrant is not only within the threshold but also *has the same or higher cost* than the point under consideration.

Note that it is possible for a plan P_J to be swallowed by another plan P_K, and later, for (the now enlarged) P_K itself to be swallowed by a plan P_M, as long as all the query points now associated with P_K do not have their original costs increased by more than the threshold.

In general, if a sufficiently large threshold is given, the user may expect to wind up with a *single* plan in the reduced plan diagram. However, in our implementation, this will happen only if the query costs follow the Cost Domination Principle mentioned above.

Abstract-plan-costing Approach

In the APC-based approach, cost domination and restricting attention to first quadrant is not mandatory since Abstract-plan-costing can be used to explicitly

evaluate the costs of replacement plans at the replaced locations. However, in our algorithms, to ensure that Abstract-plan-costing needs to be carried out at only a few points in the selectivity space, a different assumption is made with regard to the behavior of plan cost functions over the selectivity space – specifically, that all plans follow the parametrized cost model template described in this VLDB 2008 paper, which was arrived at after an in-depth study of current industrial-strength optimizers.

For a detailed treatment of the Plan Reduction problem, see this technical report (CB-based algorithms) and this technical report (APC-based algorithms).

5. Time Estimations for Exact Diagram Production

For estimating the time to produce a ***compilation*** diagram, Picasso first measures the compilation times for five query points spaced uniformly on the principal diagonal of the n-D selectivity space and scales the average of these by the total number of query points in the diagram. The above initial estimate is *re-calibrated dynamically* during the diagram processing based on the time taken for the queries fired so far and the number of remaining queries.

For estimating the time to produce an ***execution*** diagram, Picasso first measures the execution time of a query point on the principal diagonal that is located close to the origin of the n-D selectivity space. For example, in the case of a full-range diagram, the query (0.05, 0.05,**...**, 0.05). This value is used as an indicator of the multiplication factor ***f*** relating the query optimizer's estimated cost and the actual response time. Then the optimizer costs for all the points in the diagram are aggregated and finally multiplied by ***f*** to estimate the production time of the entire diagram.

The above initial estimate is *re-calibrated dynamically* during the diagram processing based on the time taken for the queries fired so far and the aggregate estimated cost of the remaining queries.

An important point to note is that since the time estimates for producing an execution

diagram are based on *actually executing* a sample query, it may therefore take significant time even to just produce the estimate. Further, the estimation could be significantly influenced by the current system environment, e.g. the contents of the database memory cache.

6. Time Estimations for Approximate Compilation Diagram Production

For estimating the time required to generate an approximate compilation diagram, coarse estimators, S-est and G-est, are incorporated in the Sampling (RS_NN) and Grid (GS_PQO) approximation algorithms, respectively.

S-est: In this time estimator for the RS_NN algorithm, a randomly-chosen small seed set of query points is initially optimized and the plan-identity error is estimated using the d_{max} statistical estimator, as described in this technical report. Then, a second randomly chosen query point set of the same size is optimized and the incremental reduction in the error is estimated. Based on this reduction, the number of samples required for reducing the error to meet the user-specified tolerance is estimated through linear scaling. Finally, the time for optimizing the expected number of required samples is estimated, again through linear scaling.

Note that our estimation process only considers plan-identity error. This is because, given similar tolerance levels for both errors, our empirical observation has been that the plan-location error typically lags behind the plan-identity error. However, it is possible that the user may have given a significantly larger tolerance for identity error as compared to location error, in which case the assumption is no longer valid. To handle this situation, we artificially treat the lower of the tolerance levels to be the plan-identity error in computing the time estimation, in the process achieving a conservative estimate (note that this assumption is only intended for generation time estimation and does not impact the approximation process itself). In the current implementation, the cardinality of the seed set is 50 query points, which typically takes less than a minute to optimize on standard platforms.

G-est: In this time estimator for the GS_PQO algorithm, the points on the corner cells of the grid-partitioned space are initially optimized. The "plan richness" factor, a

measure of the expected number of plans present in the region, is computed for each of these cells (see this technical report for details). Among the cells that are adjacent to an axis, we choose the one with the maximum plan-richness for repeated partitioning until it is estimated that the user tolerance levels have been met within the cell. Call this cell as $\mathrm{Max}_{\mathrm{axis}}$. Subsequently, the cell with the highest selectivity coordinates (i.e. the cell at the far end of the principal diagonal of the space) is also processed in a similar manner. Call this cell as $\mathrm{Max}_{\mathrm{sel}}$. Now we assume that each cell adjacent to an axis will incur the same overheads as $\mathrm{Max}_{\mathrm{axis}}$, while all remaining cells will incur that of $\mathrm{Max}_{\mathrm{sel}}$. We sum up the sample size accordingly as the final estimate. In the current implementation, the initial partitioning of the selectivity space is such that each cell has an edge length of 8 query points. The motivation for treating the cells bordering the axes differently than those in the hinterland is the commonplace observation that plans tend to be distributed richly along the axes as compared to the interior regions.

7. Execution Cost Diagram

An important point to note with regard to execution cost diagrams is that the actual running times depend on a number of factors, including system load, cache status, and also, *the order in which the query instances are generated* – this is because the order can influence the cache contents. In the current diagrams produced by Picasso, no special effort is made to produce a cold-cache – instead the diagram is a function of whatever happens to be the cache status during the diagram production process. Further, as mentioned above, the queries are generated in row order beginning with the point closest to the origin.

Note that the above issue does not arise for Execution Cardinality Diagrams, since the result cardinality is a function of only the database contents, and not the system environment.

Documentation Home

Picasso Database Query Optimizer Visualizer

TROUBLE-SHOOTING

1. If you get connection errors to the database engines, make sure that you have the appropriate database connection libraries present in the Libraries directory, and that you have executed the ***activatedb*** script.
2. If you get visualization errors, make sure that you have the appropriate graphics libraries present in the Libraries directory.
3. If the ***AutoConvert SQL Dialect*** button is greyed out, you will need to download the ***SwisSQL.API.jar*** library file into the Libraries directory (see the licenses documentation for download details). Note that this optional library is not provided even with the full distribution of Picasso.
4. If you have problems running Picasso with the Java3D installed on your system, copy the library files ***j3dcore.jar***, ***j3dutils.jar*** and ***vecmath.jar*** from the system's Java3D directory to the Libraries directory.
   In Windows, if the problem persists, edit ***runClient.bat*** as follows:
   Change *"java -Xmx256m …"* to
   "java ***–Djava.library.path=<JDK_HOME>\jre\bin*** *–Xmx256m …"*
   This directory contains ***J3D.dll*** and ***J3dUtils.dll*** which are required to run Picasso.
5. If the Picasso Client doesn't seem to be responding, or is taking a long time for some operation, check the console from which the Client was run. If there is any error message like "Out of memory exception", then close the Client and restart it. If the problem persists, increase the JVM memory amount to 1 GB by providing -Xmx1024m argument. By default it is set to 256 MB.
6. During its operations, the Picasso Client may throw an error message "Out of Memory Error, Please Restart PicassoClient". This happens when too many diagrams are created in the current session of the client. When this happens, close

the Client and start it again.

7. After clicking on a diagram tab, you may sometimes receive a black diagram screen instead of the picture. This is usually because your system does not have sufficient video memory to host all the diagrams retrieved from the server. To conserve video memory, try lowering the screen resolution and reducing the color quality. Alternatively, set the flag LOW_VIDEO in PicassoConstants.java to True (the default value is False), recompile the client, and resume operations – with this flag set to true, the video memory is flushed before each new diagram is loaded.
8. The Picasso Client window always starts in a maximized state. If it is resized to some other state during its operations, then the ***Status Bar*** may not be visible. To see the status bar, maximize the Client window.
9. The Plan Legend may not show sometimes – in such cases, try resizing the window to Normal size and then revert to Maximized size.
10. If the Picasso Client or the Picasso Server seems to hang, it could be because the associated ***console*** is in the 'Select' mode, perhaps due to a mouse click. In this situation, try to bring the console back to the normal mode by pressing the 'Enter' key in the console.
11. If the ***Machine*** field of the ***DBConnection Settings*** dialog accepts the IP of a machine, but does not accept the machine name, then add the machine to the file ***Windows\system32\drivers\etc\hosts*** | /etc/hosts in Windows | Unix and try again.
12. For a Picasso error that appears related to the database engine, first check if the query can be independently and successfully executed directly on the database engine, in order to make sure the error is not due to the query syntax or semantics (such as missing table, wrong schema, etc), and take measures accordingly.

Note: If the above solutions do not address your problem, please contact us at picasso@dsl.serc.iisc.ernet.in with a description of the problem. Both the Picasso Server and the Picasso Client write their outputs and errors into log files in the ***PicassoRun/Logs*** directory. The log files typically take up a few tens of kilobytes space for each freshly processed query template. With your email, please attach the Server and Client log files, the SelLog text file (created in Logs directory by executing Save in the File menu for the SelLog screen), as well as screen dumps of the Server and Client consoles.

KNOWN ISSUES:

1. In a Linux environment, the client may throw the exception *(<unknown>:8407): Gtk-CRITICAL **: gtk_paint_box: assertion `style->depth == gdk_drawable_get_depth (window)' failed*. This is an open bug currently being examined by Sun Developer Network. For more details, please refer to this page.

Documentation Home

Picasso Database Query Optimizer Visualizer

FUTURE PLANS

We hope to include the following features in future releases of the Picasso software:

1. Support for the Date data type in Oracle, and the String data type in PostgreSQL.
2. Knee estimators for the PlanCardinality-versus-CostIncreaseThreshold graphs of CC-SEER and LiteSEER.
3. Display of multiple diagrams side-by-side in the drawing canvas.
4. Remote applet access to Picasso installations.
5. Parallel diagram generation on multi-core systems.
6. Incremental diagram generation and partial diagram visualization.

Documentation Home

Picasso Database Query Optimizer Visualizer

CODE DESIGN and HISTORY

Design:

Refer to this master's thesis for an introduction to the overall design of Picasso, and to the Server class diagram and Client class diagram for the code organization.

History:

Version 0.5b (May 2006)

1. Limited release

Version 1.0 beta (November 2006)

1. Public release
2. Features:
    a. Rewritten Picasso parser
    b. Modular separation of code between Client and Server
3. Bugs:

No	Description	Status
1	Postgres plan parser missing right tree of binary input for Sort	Fixed in v1.0
2	Only uppercase schema information supported	Fixed in v1.0
3	Bigint type not supported for SQL Server	Fixed in v1.0
4	PlanDiff incorrectly showed some differences	Fixed in v1.0
5	Checked Cost-domination violation locally, instead of globally	Fixed in v1.0
6	Picasso parser gives error on some complex queries	Fixed in v1.0
7	Lengthy sub-operator information in SQL Server caused exception	Fixed in v1.0
8	Big-int typecasting is incorrect	Fixed in v1.0
9	Sybase ASE templates for Q17 and Q20 were incorrect	Fixed in v1.0
10	Remote connection to PostgreSQL was not working	Fixed in v1.0
11	Identical attribute names of different relations not permitted in PSPs	Fixed in v1.0

Version 1.0 (May 2007)

1. Public release
2. Major new features:
    1. Client-side implementation of Operator-based and Parameter-based plan differentiation – the user can toggle between these two views without having to create two separate diagrams
    2. Exponential distribution of query points in the selectivity space –

maximum density around the origin and along the axes, progressively lesser moving outwards in the space

3. Command-line interface to submit query templates for diagram generation in batch mode

4. CostGreedy plan reduction algorithm which is significantly more efficient than AreaGreedy, the algorithm used in the beta version

5. Abstract Plan feature through which, at a given point in the current plan diagram, the user can visualize the plan produced by another engine or by the same engine at another optimization level

6. Abstract Plan feature through which a specific plan appearing in the plan diagram can be employed through the entire selectivity space, and not just in its optimality region
(this feature is available only on MS SQL Server 2005 and Sybase ASE)

3. Minor new features:

1. Detailed operator parameter information for SQL Server

2. Added Relation nodes under index nodes for DB2 and Oracle, so that the leaf nodes are always base relations

3. Index node parameter information for Oracle

4. Improved the quality of the diagram production time estimator for execution cost diagrams by taking into account the estimated compiler costs at all points in the diagram

5. Improved the speed of the execution cost diagram time estimator by using the lower diagonal corner as the sample query point rather than the mid-point of the diagonal

6. Quantified the number of query points where Cost Domination fails

4. Other changes:

1. Several bug-fixes of version 1.0 beta listed in above table

2. Several improvements to user messages

3. Changed the term “Sub-Operator” to “Parameter” to be consistent with literature

Version 2.0 (February 2009)

5. Public release

6. Major new features:

    1. Custom Resolution for each dimension – diagram resolutions can be specified by the user on a per-dimension basis.

    2. Custom Range for each dimension – diagram production can be localized to user-specified sub-ranges along each dimension of the selectivity space.

    3. Client-side implementation of (a) color assignments to plans, and (b) slicing of diagrams for query templates with three or more dimensions.

    4. Approximate diagram generation – efficient production of approximate diagrams that are in conformance with user-specified tolerances to plan identity and plan location errors. User has a choice of two approximation algorithms: RS-NN, based on random sampling and nearest-neighbor implementation, and GS-PQO, based on grid partitioning and parametric query optimization.

    5. Robust Plan Reduction through the CC-SEER algorithm which extends the cost-increase-threshold guarantee of plan replacement to the entire selectivity space, thereby providing robustness to errors in selectivity estimates. LiteSEER, a computationally light-weight heuristic-based variant of CC-SEER is also available in the system.

7. Additional new functionalities:

    1. Multi Engine Plan View – at a given query location, side-by-side view of

plans from two different database engines, or the same engine at different optimization levels.

2. Enhanced Plan Legend Panel – for query templates with three or more dimensions, the plan legend panel for each 2-D slice displays both the global and the slice-specific number of plans. The ordering and coloring of plans in the legend is kept consistent across the slices based on global space coverage.

3. Compressed Diagram Packet – diagram packets sent from the server to the client are compressed to reduce transfer latency.

4. Multi-core Plan Operators – operator lists have been extended to support database engines featuring plan operators that are specific to multi-core platforms.

8. Other changes:

    1. Several improvements to user status and error messages

Version 2.1 (February 2011)

9. Public release

10. Major new features:

    1. MySql Support – Users can now run Picasso on MySql.

    2. Collation Schemes – Different collation schemes can now be used for PSPs which are of the String data type.

    3. Load Packet – The packets which are saved using the Save Packet feature can now be visualized in Picasso using this feature.

11. Other changes:

    1. Many bugs have been fixed and improvements to user status and error messages have been done.

Documentation Home

Picasso Database Query Optimizer Visualizer

PORTING GUIDE

This document outlines the steps for porting Picasso to database platforms other than those natively supported in the current code-base: DB2, Oracle, SQL Server, Sybase ASE, PostgreSQL.

Let the new database engine be called ***ABC.*** These are the changes that need to be carried out to port Picasso to ***ABC***:

1. Edit the file ***DBConstants.java*** in the package ***iisc.dsl.picasso.common***, adding the following lines:

   In global scope:
   import iisc.dsl.picasso.common.db.AbcInfo;

   Inside the class *DBConstants*:
   public static final String ABC = "ABC";

   Inside the initializer specification of the *databases* array:
   new AbcInfo()

2. Edit the file ***Database.java*** in the package ***iisc.dsl.picasso.server.db.***

   Add the line: *import iisc.dsl.picasso.server.db.abc.abcDatabase;*

   In the *Database* class, modify the function ***getDatabase()***, adding the following lines:

   ```
   else if(vendor.equals(DBConstants.ABC))
         db = new AbcDatabase(settings);
   ```

3. Create a package ***abc*** in the package ***iisc.dsl.picasso.server.db***

4. Create the class ***abcInfo*** (which extends ***DBInfo***) in the package ***iisc.dsl.picasso.common.db.*** In this class, define inside the constructor, the variables *name*, *defaultPort*,

treeNames, treecard and *optLevels*: (a) *name* should be the same as *DBConstants.ABC*; (b) *optLevels* is an array of Strings representing different optimization levels supported by the database; (c) *treeNames* is an array of Strings representing the names appearing in the plan tree nodes, and is used for assigning colors to the plan tree nodes; (d) *treecard* is an array of integers representing the expected number of children for corresponding nodes in *treeNames; and (e) defaultPort* is a String that denotes the default port that is shown when a new DBConnection Descriptor is created.

5. Create the class ***abcDatabase*** (which extends ***Database***) in the package ***iisc.dsl.picasso.server.db.abc***. This class should implement all the abstract functions defined in Database. They are:

abstract public boolean connect(DBSettings settings) - Should attempt to establish a connection to the given database URL (using JDBC) and return a boolean corresponding to whether or not the connection was successfully established.

abstract public Histogram getHistogram(String tabname, String schema, String attrib) - Should fill in the value and freq (frequency) Vectors of the Histogram object after reading the histogram corresponding to the requested attribute from the database.

abstract public Plan getPlan(String query) - Should create and return a Plan object corresponding to the query string after retrieving and parsing the plan got from the database engine.

abstract public Plan getPlan(String query, int startQueryNumber) - Used to set an identification number for the plan in the database's *explain* output table. In case the database engine does not support such identification or has no plan table, this function can simply call the function above.

abstract public void emptyPlanTable() - Should empty the database's plan table, if such a table exists.

abstract public void removeFromPlanTable(int qno) - Should remove the plan corresponding to the identification number passed as a parameter from the database's plan table. If no plan table exists, this function maybe empty.

abstract public boolean checkPlanTable() - Should return a boolean indicating the presence or absence of the plan table in the database, if one can exist. If the database does not have a plan table, this function should always return true.

The following functions are necessary to create the Picasso tables, whose schema is given below:

abstract protected void createQTIDMap(Statement stmt)
abstract protected void createPlanStore(Statement stmt)

abstract protected void createPlanTree(Statement stmt)
abstract protected void createPlanTreeArgs(Statement stmt)
abstract protected void createSelectivityLog(Statement stmt)
abstract protected void createSelectivityMap(Statement stmt)
abstract protected void createPicassoRangeResMap(Statement stmt)
abstract protected void createPicassoApproxSpecs(Statement stmt)
abstract protected void createPicassoColumns(Statement stmt)

1. Table *picassoqtidmap*

It stores global information about each Picasso diagram.

Column	Type	Modifiers	Description
qtid	integer	not null	Unique identifier for each Picasso Diagram generated.
qtemplate	text		Query template provided by the user for generating the Picasso Diagram.
qtname	varchar (64)	not null, unique	Unique string identifier associated with the Picasso Diagram.
resolution	integer		A value of 10,30,100,300 or 1000 indicates a diagram of the corresponding resolution on all dimensions over the entire selectivity space. A value of -1 indicates a 'custom range' or 'custom resolution' diagram.
dimension	integer		Number of dimensions of the Picasso diagram, equivalent to the number of Picasso predicates in the query template.
exectype	varchar (32)		It can have the values COMPILETIME, RUNTIME or APPROX-COMPILETIME.
distribution	varchar (32)		It can have the values UNIFORM or EXPONENTIAL.
optlevel	varchar (64)		Optimization level chosen by the user.

plandifflevel	varchar (32)		Indicates the level at which two plan trees are compared. It can have the value SUB-OPERATOR or OPERATOR.
gentime	bigint		Time of generation.
genduration	bigint		Time taken for generating the diagram.

Primary Key: (qtid)

2. Table *picassoplanstore*

It stores the plan number, compilation and execution costs, and compilation and execution cardinalities, for each point in the selectivity space.

Column	Type	Modifiers	Description
qtid	integer	not null	Unique identifier for each Picasso Diagram generated (references picassoqtidmap).
qid	integer	not null	Identifier for each point in the selectivity space ('product of resolution' number of entries).
planno	integer		Plan number corresponding to the selectivity point.
cost	double precision		Corresponding compile time estimated cost.
card	double precision		Corresponding compile time estimated cardinality.
runcost	double precision		Corresponding run time cost (0 for compile time diagrams).
runcard	double precision		Corresponding run time cardinality (0 for compile time diagrams).

Primary Key: (qtid, qid)

Foreign Key constraints: (qtid) REFERENCES picassoqtidmap(qtid) ON DELETE CASCADE

3. Table *picassoplantree*

It stores the representative plan tree for each plan appearing in a Picasso Diagram. Each record represents a node of a plan-tree.

Column	Type	Modifiers	Description
qtid	integer	not null	Unique identifier for each Picasso Diagram generated (references picassoqtidmap).
planno	integer	not null	Identifier for each distinct plan.
id	integer	not null	Identifier for each node in the plan tree.
parentid	integer	not null	Identifier of the parent node.
name	varchar (64)		String identifier associated with the node
cost	double precision		Cost corresponding to the node
card	double precision		Cardinality corresponding to the node

Primary Key: (qtid, planno, id, parentid)

Foreign Key constraints: (qtid) REFERENCES picassoqtidmap(qtid) ON DELETE CASCADE

4. Table *picassoplantreeargs*

It stores the parameter information, if any, for each node in a plan tree. There can be multiple parameters for a single node, each of them resulting in a record in the table.

Column	Type	Modifiers	Description
qtid	integer	not null	Unique identifier for each Picasso Diagram generated (references picassoqtidmap).
planno	integer	not null	Identifier for each distinct plan.
id	integer	not null	Identifier for each node in the plan tree.

argname	varchar (64)	not null	String identifier associated with the sub-operator.
argvalue	varchar (64)	not null	Value of the sub-operator.

Primary Key: (qtid, planno, id, argname, argvalue)

Foreign Key constraints: (qtid) REFERENCES picassoqtidmap(qtid) ON DELETE CASCADE

5. Table *picassoselectivitylog*

It stores for each axis point in a plan diagram, the Picasso, Plan and Predicate selectivities along with the constants used. The total number of entries inserted into the table per diagram is equal to the *sum* of the dimensional resolutions.

Column	Type	Modifiers	Description
qtid	integer	not null	Unique identifier for each Picasso Diagram generated (references picassoqtidmap).
dimension	integer	not null	Dimension number
sid	integer	not null	Identifier for the selectivity point. Ranges from 0 to the dimension's resolution - 1.
picsel	double precision		Corresponding Picasso selectivity.
plansel	double precision		Corresponding Plan selectivity.
predsel	double precision		Corresponding Predicate selectivity.
datasel	double precision		Reserved for future use.
const	varchar (512)		Constant corresponding to the selectivity.

Primary Key: (qtid, dimension, sid)

Foreign Key constraints: (qtid) REFERENCES picassoqtidmap(qtid) ON DELETE CASCADE

6. Table *picassoselectivitymap*

It maps each entry in *picassoplanstore* to entries in the *picassoselectivitylog* table.

Column	Type	Modifiers	Description
qtid	integer	not null	Unique identifier for each Picasso Diagram generated (references picassoqtidmap).
qid	integer	not null	Identifier for each point in the selectivity space ('product of resolution' times).
dimension	integer	not null	Dimension number.
sid	integer	not null	Identifier for the selectivity point. Ranges from 0 to the 'resolution in the corresponding dimension' - 1.

Primary Key: (qtid, qid, dimension)

Foreign Key constraints: (qtid) REFERENCES picassoqtidmap(qtid) ON DELETE CASCADE

7. Table *picassorangeresmap*

It stores the range and resolution information for the Picasso Diagrams which have '-1' (i.e. Custom) as their resolution value in *picassoqtidmap*.

Column	Type	Modifiers	Description
qtid	integer	not null	Unique identifier for each Picasso Diagram generated (references picassoqtidmap).
dimnum	integer	not null	Dimension number.
resolution	integer	not null	Corresponding resolution.
startpoint	double precision	not null	Corresponding start point of the selectivity range.
endpoint	double precision	not null	Corresponding end point of the selectivity range.

Primary Key: (qtid, dimnum)

Foreign Key constraints: (qtid) REFERENCES picassoqtidmap(qtid) ON DELETE CASCADE

8. Table *picassoapproxmap*

It stores information about Approximate Picasso Diagrams.

Column	Type	Modifiers	Description
qtid	integer	not null	Unique identifier for each Picasso Diagram generated (references picassoqtidmap).
samplesize	float		Sample size in percentage required for approximation.
samplingmode	integer		Specifies the approximation algorithm used. '0' for RS_NN and '1' or GS_PQO
areaerror	double precision		Specifies the area error tolerance provided by user.
identityerror	double precision		Specifies the identity error tolerance provided by user.
fpc	integer		Reserved for future use

Primary Key: (qtid)

Foreign Key constraints: (qtid) REFERENCES picassoqtidmap(qtid) ON DELETE CASCADE

9. View *picassocolumns*

It stores information about all the user tables and columns (including Picasso) present in the database.

Column	Type	Modifiers
column_name	name	
table_name	name	

owner	oid	

6. Create the class ***abcHistogram*** (which extends ***Histogram***) in the package ***iisc.dsl.picasso.server.db.abc***. The class should implement the constructor which provides an instance of abcDatabase and table name, schema and attribute name.

   abcHistogram(Database db, String tabName, String schema, String attribName)

   This class should also implement the following function which returns a constant of appropriate data-type, given a selectivity value:

   abstract public String getConstant(double sel)

Note: For information on the overall code structure of Picasso, see Code-design.

Documentation Home

Picasso Database Query Optimizer Visualizer

ALGORITHMIC DETAILS

1. Plan Reduction Algorithms

A detailed assessment of the Plan Reduction problem is available in this technical report. It is shown in the report that, given an optimality criterion of minimizing the plan cardinality of the reduced plan diagram for a user-specified cost increase threshold, finding the optimal solution is NP-Hard. Therefore, the only recourse is to develop heuristic-based algorithms – Picasso 1.0 supported two greedy algorithms, referred to as AreaGreedy and CostGreedy, respectively. In Picasso 2.0, only CostGreedy is retained since it is has significantly better performance characteristics, with regard to both reduction and efficiency, than AreaGreedy.

CostGreedy

This algorithm operates under the assumption that the Cost Domination Principle (query processing costs increase as we move outwards from the origin in the selectivity space) holds and therefore only plan swallowing possibilities in the ***first quadrant*** are considered with respect to the plan under consideration. A direct corollary of this assumption is that in exploring swallowing possibilities for a given plan, we do not need to retain all points contained in each potential swallower plan, but only those corresponding to the minimum cost in the first quadrant. The algorithm processes the query points starting from the top-right corner (with the Cost Domination assumption, this corner will have the highest cost of all points) and progressively makes its way to the origin, in the process using the Greedy SetCover algorithm to recolor the points – the complete details are in this technical report . The time complexity of CostGreedy is $O(mn)$ where n is the number of plans in the

plan diagram and m is the number of of points in the diagram, a significant improvement over AreaGreedy whose complexity is $O(m^2)$. More importantly, it can be shown that CostGreedy provides the best possible approximation factor with respect to the optimal reduction.

Knee Estimator

As an aid to help users make informed choices of the values to enter in the *Enter Cost Increase Threshold* dialog box, a rough estimate of the location of the threshold value corresponding to the "knee" in the graph characterizing "Plan Cardinality of the Reduced Plan Diagram versus Cost Increase Threshold", is provided in the box. Also provided is an estimate of the minimum threshold required to reach a desired number of plans (specified by the DESIRED_NUM_PLANS macro in PicassoConstants.java, set to 10 in the distribution) in the reduced plan diagram. The estimator implemented in Picasso is the AmmEst estimator described in this technical report.

Cost Increase Statistics

In the panel immediately to the right of the reduced diagram, we show bounds on the minimum, average and maximum cost increase (in percentage terms) computed over all points that have undergone plan replacement during the reduction process. By definition, these values will all be smaller than the user-specified cost increase threshold.

2. Robust Plan Reduction Algorithms

The above-mentioned reduction algorithms, AreaGreedy and CostGreedy, ensure that the replacement plans are within the cost-increase-threshold at all points in the optimality regions of the replaced plans. However, in practice, query optimizers are subject to significant errors in their selectivity estimates, and it may therefore be possible that the location of the query at run-time may lie outside the optimality region of the replaced plan. At such locations, the performance of the replacement plan could be much worse than the replaced plan, resulting in the replacement being harmful from a global perspective. This possibility naturally leads to the concept of a *robust* replacement – that is, a replacement where the λ-threshold criterion is satisfied at *all points in the selectivity space,* i.e. the replacement ensures global safety. Providing robust replacements requires costing of plans outside their optimality regions, and it is therefore possible only with database engines that support the *Abstract Plan Costing* feature (in Picasso, these engines are SQL

Server and Sybase ASE). Direct approaches to achieving robust plan reduction are computationally prohibitively expensive, but through empirical modeling of plan cost function behavior, we have been able to develop efficient alternatives, which are described in this technical report. The algorithm described there, called SEER, guarantees global safety and requires Abstract-plan-costing operations only along the *surfaces* of the selectivity space hyper-cube. More remarkably, we have found through detailed experiments with industrial-strength benchmark environments that SEER provides global safety *without jeopardizing anorexic reduction*. Unlike the Cost Bounding-based reduction algorithms, which are completely conditioned on Cost Domination holding true, the Abstract-plan-costing-based approach can supported a richer class of plan cost models wherein the plan cost function can have one local maxima or one local minima in the interior of the selectivity space. In Picasso 2.0, the following two variations of SEER have been implemented:

CC-SEER (CornerCube SEER)

CC-SEER guarantees global safety like SEER but improves on its efficiency by implementing a more conservative test for global robust replacement. Specifically, CC-SEER only involves Abstract-plan-costing operations at the *corner hyper-cubes* of the selectivity space and is therefore significantly faster than SEER. Moreover, its performance is resolution independent unlike SEER, and therefore the performance gap between CC-SEER and SEER increases with higher resolution diagrams. Concretely, while SEER's time complexity is $O(m.r^{d-1})$, where m is the number of points in the diagram, r is the resolution and d is the dimensionality, CC-SEER cuts it down to $O(m.4^d)$. Our experimental results indicate that CC-SEER's reduction quality is comparable to that of SEER, and it therefore provides an extremely attractive tradeoff between efficiency and reduction quality.

LiteSEER

LiteSeer is a light-weight heuristic-based variant of SEER that makes its replacement decisions solely based on Abstract-plan-costing operations at the *corners* of the hypercube, and is therefore extremely efficient. In fact, it can be shown that LiteSEER is optimal in the sense that it incurs the minimum work (complexity-wise) required by any reduction algorithm. While it does not *guarantee* global safety, our experimental results indicate that in practice, its safety and reduction characteristics are quite close to that of SEER and CC-SEER.

Knee Estimator

Currently, no estimators have been designed for CC-SEER and we intend to address this issue in our future work. As an interim measure, we suggest that users can first obtain the estimates from CostGreedy, and then use the same with CC-SEER. A word of caution, however – there may be significant differences between the Reduced-Plan-Cardinality vs. Threshold graphs of CostGreedy and CC-SEER. This is because, on the one hand, there is greater potential for reduction in CC-SEER due to using explicit costs instead of cost bounds. But, on the other hand, the global safety guarantee may prevent some replacements permitted by CostGreedy. Our experience has been that the latter factor dominates in the higher-resolution diagrams and therefore the knee of the graph is typically at a larger threshold for CC-SEER as compared to CostGreedy.
The above-mentioned issues for CC-SEER arise with LiteSEER as well, and due to its relaxed safety criterion, LiteSEER's knee will always be lower than that of CC-SEER.

Cost Increase Statistics

The cost increase statistics, mentioned above for the CostGreedy and AreaGreedy algorithms, are not provided for the robust reduction algorithms, CC-SEER and LiteSEER. This is because computing these statistics would require costing (using Abstract-plan-costing) of the replacement plans at all replaced points in the selectivity space, which would be prohibitively expensive. However, at any specific replaced point in the reduced plan diagram, the cost of the replacement plan can be ascertained by using the *Shift+Right-click* command which provides the ***QueryInfo*** message box.

3. Plan Difference Algorithm

The plan difference algorithm visually highlights the difference between a pair of selected plans. The common parts between the two plan trees are shown in white, whereas all differences are colored as mentioned in the Plan Tree Windows documentation, reproduced below:

§ Node labels, Node parameters and Node inputs are identical: White nodes with Black input links

§ Node labels are the same: White Fill

- Parameters different: Green-bordered nodes with Black input links
- Left and right inputs swapped: Orange-bordered nodes with Blue input links
- Left and right inputs different: Orange-bordered nodes with Red input links

§ Node labels are different: Red-bordered nodes filled with the native node color

- Left and right inputs swapped: Blue input links
- Left and right inputs different: Red input links

§ Nodes are unmapped (i.e. no corresponding node in the other tree): Nodes filled with the native node color and Black input links

We describe here the node matching process between the two trees, $\boldsymbol{P_A}$ and $\boldsymbol{P_B}$. In our description, the term “branch” is used to refer to any connected chain of nodes between a pair of branching nodes, or between a branching node and a leaf, in these trees. Branches are directed from the lower node to the higher node. The matching proceeds as follows:

1. *First all the leaf nodes (relations) and all the nodes with binary inputs (typically join nodes) are identified for P_A and P_B.*

2. *A leaf of P_A is matched with a leaf of P_B if and only if they both have the same relation label. In the situation that there are multiple matches available (that is, if the same relation label appears in multiple leaves), an edit-distance computation is made between the branches of all pairs of matching leaves between P_A and P_B. The assignments are then made in increasing order of edit-distances.*

3. *A binary node of P_A is matched with a binary node of P_B if the set of base*

relations that are processed is the same. If the node label, node parameters, and the left and right inputs are identical (in terms of base relations), the node is colored white and the input links are black. However, if the node label is different, or the node parameters are different, or if the left and right input relation subsets are different, then the node and the input links are colored as per the above coloring scheme.
***4.** A minimal edit-distance computation is made between the branches arising out of each pair of matched nodes, and the nodes that have to be added or deleted, if any, in order to make the branches identical are colored as per the above coloring scheme. Unmodified nodes, on the other hand, are matched with their counterparts in the sibling tree and colored either white or white with green border.*

Note: When plan difference is carried out on a ***OperatorDiff***-based plan diagram, the treeswill always include at least one colored node or link (that is, unlike ***ParameterDiff -based*** plan diagrams, it is not possible to have pure black-and-white trees with only green node borders).

4. Approximation Algorithms

Plan diagrams can be computationally extremely expensive for plan diagrams on higher-dimension query templates and fine-grained diagram resolutions. For example, a 3D plan diagram with a resolution of 100 on each dimension could well take about a week to produce. To address this problem, diagram approximation algorithms have been incorporated in Picasso 2.0. Specifically, these algorithms produce approximate diagrams by explicitly optimizing only a small subset of points from the query selectivity space and *inferring* the remaining points, thereby drastically reducing the computation time. Two categories of errors can arise in the approximation process:

1. Plan Identity Error: This error metric refers to the possibility of the approximation missing out on a subset of the plans present in the exact plan diagram. It is computed as the percentage of plans lost in the approximation process. This error is challenging to control since a majority of the plans appearing in plan diagrams are typically small in area and therefore hard to find.
2. Plan Location Error: This error metric refers to the possibility of incorrectly assigning plans to query points in the approximate plan diagram. It

is computed as the percentage of incorrectly assigned points. This error is also challenging to control since the plan boundaries can be highly non-linear and are sometimes even irregular in shape.

We take from the user the tolerance levels for these two errors and our algorithms attempt to ensure that the actual errors in the approximate diagram are in the close proximity of these thresholds, while at the same time minimizing the diagram production overheads. The two approximation algorithms currently implemented in Picasso are briefly described below.

Random Sampling with Nearest Neighbor Inference (RS_NN)

This algorithm employs classical random sampling and nearest neighbor classification to generate approximate diagrams. It begins by optimizing a small seed-set of randomly chosen points in the diagram, and then repeatedly optimizes a similarly-sized random group from the remaining un-optimized query points in each successive iteration. The stopping condition for the iteration is when the identity-error is expected, using a statistical estimator, to have reached below the user's tolerance level. After this, the algorithm repeatedly (a) infers the plan assignments for all un-optimized points using nearest neighbor classification, and then (b) explicitly optimizes a subset of these points, until the location-error is estimated to have reached below the user's tolerance level. To reduce the number of misclassifications along plan region boundaries, the plan assignments of the inferred points in the diagram are passed through a low-pass filter in the final step.

Grid Partitioning with Parametric Query Optimization Inference (GS_PQO)

This algorithm employs grid partitioning and the basic notions of parametric query optimization, along with plan-tree differencing techniques to generate approximate diagrams. Unlike RS_NN, heuristics are used to measure the expected quality of approximation. The entire diagram is partitioned into a coarse grid and the plan richness factor is computed for each of the grid cells. The plan richness is determined by the extent of structural differences between the plans at the corners of the cell. The cells are first arranged in a max-heap structure based on their plan richness factor. Then, the cell at the top of the heap is extracted and repeatedly sub-divided until the plan richness of the resultant cells reaches below a certain threshold (this value is derived directly from the user given error tolerances).

The plan assignment process operates as follows: *If any pair of adjacent corner points have different plans, then explicitly optimize the middle point of the edge joining these two corners, otherwise assign the corner plan to the middle point. Then, process the center point of the cell based on the plans lying on the cross-hairs joining the middle points. Break the cell into smaller cells using the newly assigned points and calculate their plan richness factor. Insert them into the heap.*

Once all the rectangles in the heap reach a plan richness factor below the desired threshold, the plans are inferred for all the remaining un-optimized points — the middle point of each edge is assigned a plan chosen randomly from the plans at the corners of the edge. Finally, the algorithm terminates if the heap is empty.

For detailed descriptions of the approximation algorithms and the associated experimental results, please refer to this technical report .

Documentation Home

Picasso Database Query Optimizer Visualizer

PUBLICATIONS

PAPERS

- Analyzing Plan Diagrams of Database Query Optimizers
  Naveen Reddy and Jayant Haritsa
  Proc. of 31st Intl. Conf. on Very Large Data Bases (VLDB), Trondheim, Norway, September 2005.

- On the Production of Anorexic Plan Diagrams
  Harish D., Pooja Darera and Jayant Haritsa
  Proc. of 33rd Intl. Conf. on Very Large Data Bases (VLDB), Vienna, Austria, September 2007.

- Identifying Robust Plans through Plan Diagram Reduction
  Harish D., Pooja Darera and Jayant Haritsa
  Proc. of 34th Intl. Conf. on Very Large Data Bases (VLDB), Auckland, New Zealand, August 2008.

- Efficiently Approximating Query Optimizer Plan Diagrams
  Atreyee Dey, Sourjya Bhaumik, Harish D. and Jayant Haritsa
  Proc. of 34th Intl. Conf. on Very Large Data Bases (VLDB), Auckland, New Zealand, August 2008.

- On the Stability of Plan Costs and the Costs of Plan Stability
  M. Abhirama, Sourjya Bhaumik, Atreyee Dey, Harsh Shrimal and Jayant Haritsa
  Proc. of 36th Intl. Conf. on Very Large Data Bases

(VLDB), Singapore, September 2010.

- The Picasso Database Query Optimizer Visualizer
  Jayant Haritsa
  Proc. of 36th Intl. Conf. on Very Large Data Bases (VLDB), Singapore, September 2010.

- ## THESES

- Next Generation Relational Query Optimizers
  Naveen Reddy
  ME Thesis, Dept. of Computer Science and Automation, Indian Institute of Science, June 2005.

- Picasso: Design and Implementation of a Database Optimizer Analyzer
  Mohammed Aslam
  ME Thesis, Dept. of Computer Science and Automation, Indian Institute of Science, July 2006.

- Picasso: Analyzing and Characterizing Relational Query Optimizers
  Akshat Nair
  ME Thesis, Dept. of Computer Science and Automation, Indian Institute of Science, July 2006.

- Picasso 1.0: Design and Analysis
  Tarun Ramsinghani
  ME Thesis, Dept. of Computer Science and Automation, Indian Institute of Science, July 2007.

- Reduction of Query Optimizer Plan Diagrams
  Pooja Darera
  MS Thesis, Supercomputer Education & Research Center, Indian Institute of Science, December 2007.

- SIGHT and SEER: Efficient Production and Reduction of Query Optimizer Plan Diagrams
  Harish D
  ME Thesis, Dept. of Computer Science and Automation, Indian Institute of Science, July 2008.

- Efficient Generation of Query Optimizer Plan Diagrams
  Sourjya Bhaumik
  ME Thesis, Dept. of Computer Science and Automation, Indian Institute of Science, June 2009.

- Characterizing Plan Diagram Reduction Quality and Efficiency
  Harsh Shrimal
  ME Thesis, Dept. of Computer Science and Automation, Indian Institute of Science, June 2009.

- Design and Implementation of Picasso 2.0
  Ravi Shetye
  ME Thesis, Dept. of Computer Science and Automation, Indian Institute of Science, July 2009.

- On the Stability of Plan Costs and the Costs of Plan Stability
  M. Abhirama
  ME Thesis, Dept. of Computer Science and Automation, Indian Institute of Science, August 2009.

- Efficiently Approximating Query Optimizer Diagrams
  Atreyee Dey
  MS Thesis, Supercomputer Education & Research Centre, Indian Institute of Science, August 2009.

- TECHNICAL REPORTS

- Reduction of Query Optimizer Plan Diagrams
  Harish D., P. Darera and J. Haritsa
  Technical Report TR-2007-01, DSL/SERC, Indian Institute of Science, March 2007.

- Robust Plans through Plan Diagram Reduction
  Harish D., P. Darera and J. Haritsa
  Technical Report TR-2007-02, DSL/SERC, Indian Institute of Science, November 2007.

- Efficient Generation of Approximate Plan Diagrams
  A. Dey, S. Bhaumik, Harish D. and J. Haritsa
  Technical Report TR-2008-01, DSL/SERC, Indian Institute of Science, March 2008.

- Stability-conscious Query Optimization
  M. Abhirama, Sourjya Bhaumik, Atreyee Dey, Harsh Shrimal and Jayant Haritsa
  Technical Report TR-2009-01, DSL/SERC, Indian Institute of Science, October 2009.

Documentation Home

Picasso Database Query Optimizer Visualizer

LICENSE INFORMATION

1. Picasso Software

The Picasso Database Query Optimizer Visualizer software is copyrighted by the Indian Institute of Science, Bangalore, India, as per this Copyright Certificate issued by Govt. of India, on May 1, 2006. The entire software package, including the source code, is provided ***completely free of charge*** on an *as-is-where-is* basis. Downloading the Picasso software automatically implies that you accept and acknowledge copyright ownership by the Indian Institute of Science, and that your usage of the Picasso software is governed by the terms of this licensing agreement .

2. Third-Party Software

Third-party software required for Picasso to function is comprised of libraries for
(a) graphics layout and visualization
(b) establishment of connections with the various database engines
(c) SQL query dialect conversion, and
(d) scientific computations

All rights on these support libraries rest entirely with the respective vendors.

To assist in setup and testing of your initial Picasso installation, all essential libraries are included with the full version of the Picasso code-base in the Libraries folder – however, we *strongly recommend* that users should visit the vendor URLs given below for complete details about the licensing of the support libraries, and make sure

that their usage complies with the vendor's requirements. Users are fully responsible for their usage of the support software and Indian Institute of Science is not liable for any improper usage. As a general policy, it is advisable for Picasso users to directly download the support libraries from the vendor web-sites, or from alternative third-party sources, and update the Libraries folder with these downloaded files.

2.1 Graphics

Java3D: This is an extension to Java for displaying three dimensional graphics, distributed under BSD and JRL licenses, resdistributable as per this document.
Libraries used: *j3dcore.jar j3dutils.jar vecmath.jar*

VisAD: This is a Java component library for interactive and collaborative visualization and analysis of numerical data, distributed under an LGPL license, redistributable as per this document.
Libraries used: *visad.jar*

JGraph: This is a Java library for the visualization and layout of graphs. While the basic JGraph library is freely available, Picasso also makes use of the commercial JGraph Layout library which we have purchased, and is redistributable as per this licensing agreement.
Libraries used: *jgraph.jar jgraphlayout.jar*

Swing-Layout: This is a Java component library for layout of graphic components, distributed under an LGPL license, redistributable as per this document.
Libraries used: *swing-layout-1.0.jar*

2.2 Databases

DB2 JDBC Driver: This is a JDBC driver for connecting to DB2 databases, redistributable as per this document.
Libraries used: *db2jcc.jar db2jcc_license_cu.jar*

Oracle JDBC Driver: This is a JDBC driver for connecting to Oracle databases,

redistributable as per this licensing agreement.
Libraries used: *ojdbc14.jar*

MSSQL JDBC Driver 2.0: This is a JDBC driver for connecting to Microsoft SQL Server, redistributable as per this licensing agreement.
Libraries used: *sqljdbc4.jar*

Sybase ASE JDBC Driver: This is a JDBC driver for connecting to Sybase ASE database products and its use is as per the terms given in this document – we have received permission from Sybase ASE for redistribution of the driver.
Libraries used: *jconn3.jar*
Picasso also uses a *hist_values* stored procedure, written by Kevin Sherlock, for reading the statistical summaries in Sybase ASE.

PostgreSQL JDBC Driver: This is a JDBC driver for connecting to PostgreSQL, distributed under a BSD license, redistributable as per this document.
Libraries used: *postgresql-8.0-311.jdbc3.jar*.

MySQL JDBC Driver: This is a JDBC driver for connecting to MySQL, redistributable as per this document.
Libraries used: *mysql-connector-java-5.1.8-bin.jar.*

SwisSQL API (Java): This is a multi-dialect SQL parser and conversion tool, which supports the automatic conversion of SQL queries across database engine dialects. The tool can be obtained through email enquiry. It is available for payment under a commercial license, and free of charge under an academic license. A trial version is also available for free download. [Note: This optional library is not included in the Picasso distribution.]
Libraries used: *SwisSQLAPI.jar*

Driver Sites:

Comprehensive lists of database drivers, including both vendor drivers and third-party drivers, are available at the following sites:

- Sun: http://developers.sun.com/product/jdbc/drivers
- Minq Software: http://www.minq.se/products/dbvis/drivers.html

2.3 Scientific Operations

LinearAlgebra: This is a Java library for linear algebra operations, produced by the Statistics group of the Forest Product Laboratory, established by the U.S. Department of Agriculture Forest Service. It is redistributable as per this document. In the Picasso distribution, for ease of packaging and installation, we have constructed from the original library a restricted pic_linearalgebra.jar library comprised only of the specific files mentioned below. [Note: This library is included in both the ***full*** and ***no-lib*** versions of the Picasso distribution.]
Files used: *BLAS_j.java, QR_j.java, LU_j.java, NotFullException.java*

Documentation Home

Picasso Database Query Optimizer Visualizer

DEVELOPMENT TEAM

- Jayant Haritsa (Project Lead)

(primary student contributors in chronological order of participation)

- Naveen Reddy (ME, CSA, IISc)
- Mohammed Aslam (ME, CSA, IISc)
- Akshat Nair (ME, CSA, IISc)
- Vidya Bharat (Project Associate)
- Shruthi A (MS, SERC, IISc)
- Tarun Ramsinghani (ME, CSA, IISc)
- Pooja Darera (MS, SERC, IISc)
- Abhijit Pai (ME, CSA, IISc)
- Harish D (ME, CSA, IISc)
- Sourjya Bhaumik (ME, CSA, IISc)
- Atreyee Dey (MS, SERC, IISc)
- Ravi Shetye (ME, CSA, IISc)
- M. Abhirama (ME, CSA, IISc)
- Harsh Shrimal (ME, CSA, IISc)
- Anshuman Dutt (ME, CSA, IISc)
- Mahesh Bale (ME, CSA, IISc)
- Mayuresh Kunjir (ME, CSA, IISc)
- Priyank Mehta (PA)
- Rakshit Trivedi (PA)
- Bruhathi HS (MS, SERC, IISc)

Documentation Home

Picasso Database Query Optimizer Visualizer

ACKNOWLEDGEMENTS

- This work was supported in part by a Swarnajayanti Fellowship from the Dept. of Science & Technology, Govt. of India, by a research grant from the Dept. of Bio-technology, Govt. of India, and by generous grants from our industrial benefactors including IBM, Google, Microsoft and Sybase.

- We thank the faculty, students and staff of the Supercomputer Education & Research Centre and the Dept. of Computer Science & Automation at the Indian Institute of Science for their sustained support and encouragement of our activities over the years.

Documentation Home

Picasso Database Query Optimizer Visualizer

TPC-H data generation and loading

Note: Picasso can be used with any generic relational database schema and SQL queries. The examples in the Picasso documentation are with respect to the TPC-H benchmark, and the procedure for generating and loading the TPC-H database is given here.

1. Download the TPC-H benchmark programs DBGEN and QGEN from the TPC-H site.

2. Create the TPC-H schema in your database engine using the file ***dss.ddl*** as per these instructions: DB2 Oracle SQL Server Sybase ASE PostgreSQL MySql

3. Generate the data.

    a. Compile DBGEN and QGEN after creating a suitable ***makefile***.
    b. Run it to produce the data files (***.tbl*** files) and queries.

4. Load the TPC-H data as per these instructions: DB2 Oracle SQL Server Sybase ASE PostgreSQL MySql

Note: While loading data into tables, it is preferable to initially ***not*** specify the integrity constraints (*primary key*, *foreign key* and *index)* since it might slow down the insertion process. These constraints can be added later as mentioned below after the data has been loaded into the tables.

5. Apply the integrity constraints on the data using file ***dss.ri*** as per these

instructions: DB2 Oracle SQL Server Sybase ASE PostgreSQL MySql

Note 1: The ***dss.ddl*** and ***dss.ri*** files may have to be modified to suit the syntax requirements of particular database engines. For example:

a. The ***connect to*** statement should not be present for Oracle. Also this statement is not required (for other engines) if a connection to the database has already been established. The equivalent command for Sybase ASE (through the ***isql*** prompt) is ***use <dbname>***. And for PostgreSQL (through the ***psql*** prompt), it is *\c <dbname>*

b. Sybase ASE (through the ***Interactive SQL*** prompt) requires that the script should not have a semi-colon (;) at the end of any statement.

c. The schema name (qualifier to the table names) should be changed to reflect the actual schema name or the table names should be made un-qualified.

d. While specifying foreign key constraints, Oracle requires the keyword ***constraint*** and the constraint name to appear *before* the keywords ***foreign key***.

e. While specifying any constraints, SQL Server, Sybase ASE and PostgreSQL require that these constraints should *not be named.*

f. The ***isql*** prompt of Sybase ASE requires that there should be *no* semi-colon after each statement – instead there should be a ***go*** command.

g. The ***commit work*** statements are not needed since any *ddl* command is generally automatically followed by a *commit*. In the case of some engines, this could also lead to a syntax error.

h. SQLServer uses the ***datetime*** data type in place of ***date***.

Note 2: The *.**tbl*** file may have to be modified to suit the syntax requirements of particular database engines. For example,

a. PostgreSQL requires that the delimiter should *not* appear at the end of every line and that string values should *not* be quoted.

b. DB2 requires that the values should be comma-separated and the string values should be enclosed in double quotes (“”).

Documentation Home

Picasso Database Query Optimizer Visualizer

DB2

(The following setup procedure is known to work with IBM DB2 v8, v9 on Windows.)

1. Install DB2 with complete support for Java
   a. Run ***setup.exe*** and follow the wizard.
   b. In the ***Configure DB2 Instances*** screen, you can customize the Protocol settings (including port number and service name) and Startup settings of the instance.

2. If the DB2 service is not already running, start it with the command ***db2start***.

3. Open a command window using Start → Programs → IBM DB2 → Command Line Tools → Command Window

4. Create a database by giving the command: ***db2 create database <dbname>***.
Connect to it using the command: ***db2 connect to <dbname>***

5. Create the schema. If the schema to be created is in a file, run the command: ***db2 –tvf <filename>***

6. Load the data into the database tables. If you are loading data from a file, use this command:

   db2 import from <filename> of del commitcount <value> insert into <tablename>

For example,
db2 import from nation.tbl of del commitcount 100 insert into NATION

7. Create the integrity constraints (primary key, foreign key and indexes) on the data. If statements are in a file, run the command: ***db2 -tvf <filename>***

8. Create the ***explain plan*** tables. These are the tables in which query execution plans are stored by the optimizer. To create these tables, after connecting to the database, run the following command in the location \IBM \SQLLIB\MISC:
db2 -tvf EXPLAIN.DDL

9. Create ***statistical summaries*** for all table columns that may be used as Picasso predicates in the query templates, with the command:
db2 runstats on table <fully qualified table name> on columns (column_list) with distribution

For example, if statistics have to be collected on the columns N_NAME and N_COMMENT of the table NATION belonging to schema 'admin', the command is:
db2 runstats on table admin.NATION on columns (N_NAME, N_COMMENT) with distribution

To improve the scope for optimization, you could optionally create statistics for *all* columns of each table. This can be done using the command:
db2 runstats on table <fully qualified table name> with distribution on all columns
where
with distribution specifies that distribution statistics are requested, and
all columns updates statistics on all the columns of the table.

Note:
1. The diagrams on the Picasso website have been generated with statistical

summaries created on ***all*** table columns.
2. With recent versions of DB2 (e.g. v9.5), most of the above-mentioned operations can be conveniently performed through the DB2 Control Center GUI.
3. The ***explain plan*** tables and the data tables should be created under the same schema name.

10. Optionally, run the following commands to enhance the scope for optimization:

db2set db2_extended_optimization=on
db2 update database cfg for tpch using sortheap <memory allocation>

Documentation Home

Picasso Database Query Optimizer Visualizer

Oracle

(The following setup procedure is known to work with Oracle 9i, 10g, 11g on Windows.)

1. Install Oracle: Run ***setup.exe*** and follow the wizard.

2. Open the SQL Plus command prompt window through one of the following ways:

    a. Start → Programs → Oracle → OraHome → Application Development → SQL Plus. Login as system/manager (or any other valid user/password).
    b. Type ***sqlplus system/manager*** (or any other valid user name and password) in the command prompt.

3. Create the schema. If the schema to be created is in a file, run the following command in the SQL prompt:

    @ <full path>\<filename>

4. Load the data into the database tables. If you are loading data from a file:

    a. Create a ***.ctl*** file for each table with the following contents (in the same location as the data files):

```
load data
INFILE '<filename>'
INTO TABLE <tablename>
FIELDS TERMINATED BY '<delimiter>'
<table_format>
```

    For example, if the data is pipe-separated, the ***customer.ctl*** file will have these contents:

```
load data
```

```
INFILE 'customer.tbl'
INTO TABLE  CUSTOMER
FIELDS TERMINATED BY '|'
(C_CUSTKEY, C_NAME, C_ADDRESS, C_NATIONKEY,
C_PHONE, C_ACCTBAL, C_MKTSEGMENT,
C_COMMENT)
```

b. After creating this file, open a command prompt, navigate to the location of the above files and run the command:

sqlldr <username>/<password> control = <filename>

For example, for username 'system' loading table CUSTOMER with files located in C:\, navigate to C:\ and run the command:

sqlldr system/manager control = customer.ctl

5. Create the integrity constraints (primary key, foreign key and indexes) on the data. If you are using a file for this, run the following command in the SQL prompt:

@ <full path>\<filename>

6. Create the ***explain plan*** tables. These are the tables in which query execution plans are stored by the optimizer. To create these tables, use this command in the SQL prompt:

@ <path of oracle home>\rdbms\admin\utlxplan.sql

For more information about Oracle plan tables, see http://www.adp-gmbh.ch/ora/explainplan.html

7. Create ***statistical summaries*** for all table columns that may be used as Picasso predicates in the query templates, using the following command in the SQL prompt:

EXEC dbms_stats.gather_table_stats ('<schema_name>', '<table_name>', method_opt => 'FOR COLUMNS <column_names>');

For example, to create statistics on the N_NAME and N_COMMENT columns of the NATION table:

EXEC dbms_stats.gather_table_stats ('system', 'NATION', method_opt => 'FOR COLUMNS N_NAME,

N_COMMENT');

To improve the scope for optimization, you could optionally create statistics for *all* columns of each table. This can be done using the command:

EXEC dbms_stats.gather_schema_stats ('<schema_name>', method_opt => 'FOR ALL COLUMNS', cascade => TRUE);

Note: The diagrams on the Picasso website have been generated with statistical summaries created on ***all*** table columns.

8. You can optionally set the following parameters to fine-tune the optimization environment (the diagrams on the Picasso website were generated using the default values for all these parameters):

    a. query_rewrite_enabled
    b. optimizer_dynamic_sampling
    c. optimizer_mode
    d. optimizer_features_enabled

    For a more exhaustive list of optimization parameters, see http://www.dba-oracle.com/art_ault_optimization_parameters.htm

Notes:

1. Creating statistical summaries is computationally intensive and may take significant time to complete. If there is an error "unable to extend temp segment in tablespace <tablespace_name>", enable the expansion of the datafile in that tablespace by checking the '*Automatically extend datafile when full*' check box in the datafile properties. Then execute the above commands again.

2. For more information on gathering Oracle statistics, see: http://www.psoug.org/reference/dbms_stats.html

Documentation Home

Picasso Database Query Optimizer Visualizer

SQL Server

(The following setup procedure is known to work with Microsoft SQL Server 2000, 2005, 2008 on Windows.)

1. Install MS SQL Server. Run ***setup.exe*** (or ***Autorun.exe***) and follow the wizard for installation of MS SQL Server. Install with SQLServer authentication. You will be asked for the password for username "sa".

2. Open Enterprise Manager using Start → Programs → Microsoft SQL Server → Enterprise Manager.

3. Create a new database by right-clicking the Databases and selecting New Database in the tree view in the left pane.

4. Create the tables using the Enterprise Manager. Under the database you just created, right-click on Tables → New Table. Here you can enter the table information.

5. Load the data into the database. If the data to be loaded is in files, right click on the database and select Tasks → Import Data and follow the wizard. Select the data source as Text File and follow the wizard to completion.

6. Create primary key, foreign key and indexes. In the Enterprise Manager, navigate to the table and right-click on it and select Design. Here you can add keys, indexes, constraints etc.

7. Create ***statistical summaries*** for all table columns that may be used as Picasso predicates in the query templates, with this command:
   CREATE STATISTICS <stats_name> ON <table_name>

(column_list) WITH FULLSCAN

To improve the scope for optimization, you could optionally create statistics for ***all*** columns of each table of the database using the command:

exec sp_createstats

Note: The diagrams on the Picasso website have been generated with statistical summaries created on ***all*** table columns.

Documentation Home

Picasso Database Query Optimizer Visualizer

Sybase ASE

(The following setup procedure is known to work with Sybase ASE 15 on Windows.)

1. Install Sybase by running ***setup.exe*** and following the wizard.

2. Create a device (or two) for the new database and allocate enough space on the devices. Create a database using the ***Add Database*** utility of ***Sybase Central*** and allocate these new devices to this database.

3. Open an SQL prompt in one of the following two ways:

    1. Open the ***Interactive SQL*** utility of Sybase and select the database name from the drop-down list of databases.
    2. In the command prompt, type ***isql –S <servername> –U <username> -P <password>*** and in this prompt, give the command ***use <dbname>*** followed by a ***go*** command. The *servername* is the name of the machine where this DB service is running.
    The ***isql*** command file is usually found in the ***Sybase\OCS\bin*** directory.
    This prompt requires that there should be *no* semi-colon after a statement, and instead there should be a ***go*** command following each statement.

4. Create the stored procedure ***hist_values*** to obtain the information required to create statistics.

5. Create the tables. If the schema to be created is in a file, use one of the following methods depending on which of the above SQL prompts is being used:

   a. In the Interactive SQL prompt, go to File → Run Script and select the required file to run.
   b. In the command prompt, type ***isql –S <servername> –U <username> -P <password> -i <filename>*** to run the file. The file should have *no* semi-colon after a statement, and instead, there should be a ***go*** command after each statement.

6. Load the data into the database tables. If the data to be loaded is in files, any of these utilities can be used:

   a. ***Data → Import*** menu item of the ***Interactive SQL*** window. (This utility requires the data files to be named with a '.csv' extension.)

   b. The ***bcp*** utility. To enable this, in the ***Sybase Central*** utility, navigate to the database name in the tree view, right-click on it and select ***Properties***. In the Options tab, check the "select into/bulkcopy/pllsort" item. Then run the following command in the command prompt:

      bcp <dbname>.<schema>.<tablename> in <filename> -c –t "<delimiter>" –r "<return_character>" -U<user> -P<passwd>
      For example, if the file *nation.tbl* has pipe-delimited data with a pipe at the end of each line, and the database is ***tpch***, schema is ***dbo,*** user is ***sa***/ ***sysadmin***, and table is ***NATION***, the command looks like this:
      bcp tpch.dbo.NATION in nation.tbl -c –t "\|" –r "\|\n" -Usa -Psysadmin

7. Create the integrity constraints (primary key, foreign key and indexes) on the data. If you are using a file for this, run it similar to step 5 above.

8. Create ***statistical summaries*** for all table columns that may be used as

Picasso predicates in the query templates. These summaries can be created using the following command in the SQL prompt:

update statistics <db_name>.<schema_name>.<table_name> (<column_name>)

To improve the scope for optimization, you could optionally create statistics for *all* columns of each table. This can be done using the command:

update all statistics <table_name>

Note: The diagrams on the Picasso website have been generated with statistical summaries created on ***all*** table columns.

Notes:

1. If the transaction logs become full, they can be cleared using the command:

   dump tran <dbname> with no_log

2. If you get an error like “There is not enough memory in the procedure cache”, then restart the Sybase service and increase the procedure cache size by giving the command

   sp_configure ‘procedure cache’, <new_size>

   The original procedure cache size can be viewed using the command:

   sp_configure ‘procedure cache’

Documentation Home

Picasso Database Query Optimizer Visualizer

PostgreSQL

(The following setup procedure is known to work with PostgreSQL 8 on Windows.)

1. Install PostgreSQL by running the .msi installer file. The process will include creating two accounts: a service account (preferably called ***postgres***) and an internal database account. The service account is an OS user account, which will be created during this installation if it doesn't exist. It should not be a privileged (root or admin) user.

2. Grant full permissions to this service account on the PostgreSQL installation folder.

3. Login to the machine as the PostgreSQL service account. The database is already initialized during installation. You can optionally initialize a new instance (in a new data folder) by running this command in the command prompt:
   initdb –D <data_folder name with path>
   For example, if the data folder is 'data1' in the current directory, the command is:
   initdb –D data1

4. Start the service with the command:
   postmaster –D <data_folder name with path>
   For example,
   postmaster –D data1

5. Create a database that will contain the database tables. The command is:
   createdb <dbname>

6. Open the PostgreSQL prompt with the command:
   psql <dbname> -U <db_username>

   Here, the db_username should be the internal database account. You will be prompted for the password, after which the SQL prompt opens.

   Note: If the ***–U <db_username>*** option is not specified, this command assumes the current OS user as the db user, and prompts for the password. Incidentally if the current OS user name is the same as the internal database account, you will be directly logged in.

7. Create the tables by entering appropriate SQL statements. If the schema to be created is in a file, run the command: ***\i <filename>*** in the psql prompt.

8. For loading tables into the database, use the ***COPY*** utility of PostgreSQL. The command template is given below. Run the command in the psql prompt.
   COPY <tablename> FROM '<filename>' WITH DELIMITER AS '<delimiter>';

   The filename should be given with the full absolute path.

   For example, to load data file C:\nation.tbl which has pipe-separated data, into the NATION table, the command is:
   COPY NATION FROM 'C:\\nation.tbl ' WITH DELIMITER AS '|';

   Note: Before loading large data files, it may be helpful w.r.t. response time to set the '***checkpoint_segments***' to a higher value than the default in the file ***postgresql.conf*** in the data directory.

9. Create the integrity constraints (primary key, foreign key and indexes) now. If the commands are in a file, run the command: ***\i <filename>*** in the psql prompt.

10. Create ***statistical summaries*** for each of the columns that may be used as Picasso predicates in the query templates using the following command in the psql prompt:

VACUUM ANALYZE;

Note: The diagrams on the Picasso web-site have been generated with statistical summaries created on ***all*** table columns.

Documentation Home

Picasso Database Query Optimizer Visualizer

MySQL

(The following setup procedure is known to work with MySQL 5.1, 5.4.1 and 5.5.9 on Windows with InnoDB as storage engine.)

1. Install MySQL: Run ***setup.exe*** and follow the wizard.

2. Create the schema. If the schema to be created is in a file, run the following command on the command prompt:

   mysql --host=localhost -u UserName -p Password --database=DbName < "filename"

   For example,

   mysql --host=localhost -u root -p pass123 --database=TPCH < "c:\\tpch\\dss.ddl"

   Note: escape sequence "\\" required on windows while giving file path.

3. Load the data into the database tables. If you are loading data from a file, run the following command after opening MySQL Command Prompt as mentioned in above step with --host –u and --database options:

```
load data local
INFILE '<filename>'
INTO TABLE '<tablename>'
FIELDS TERMINATED BY '<delimiter>'
```

For example, if the data is pipe-separated:

```
load data local
INFILE 'customer.tbl'
INTO TABLE  CUSTOMER
FIELDS TERMINATED BY '|'
```

4. Create the integrity constraints (primary key, foreign key and indexes) on the data. If you are using a file for this, run the following command in the command prompt:

   mysql --host=localhost -u UserName -p Password --port=portno --database=DbName < "filename"

   For example,

   mysql --host=localhost -u root -p pass123 --port=3308 --database=TPCH < "c:\\tpch\\dss.ri"

5. Whenever the data is changed, update statistics for tables that may participate in the query templates, using the following command in the MySQL Command Line Client:

   ANALYZE TABLE table_name;

6. You can optionally set the following parameters to fine-tune the optimization environment (the diagrams on the Picasso website were generated using the default values for all these parameters):

   a) optimizer_search_depth

   b) optimizer_prune_level

For detailed explanation of optimization parameters, see

http://dev.mysql.com/doc/refman/5.5/en/controlling-query-plan-evaluation.html

Notes:

1. PSP should be on an indexed attribute.

2. Query Template should not involve any form of sub-query. Valid query templates include SPJ queries which may contain aggregation functions, GROUP BY, ORDER BY and HAVING clause.

3. For compiled plan tree diagram, cost for intermediate nodes is not available. Only final cost is available.

4. As the histograms are not natively supported by the database, we have used the equi-depth and value based techniques to construct them. The type of histogram depends on the number of distinct values. If the distinct values are fewer than number of buckets (default 75), a value based histogram is constructed otherwise an equi-depth histogram is constructed.

5. The diagrams on the Picasso website have been generated with index on the varying attribute and InnoDB as the storage engine.

Documentation Home