BasicQuery

Taken from a presentation to the CDJDN covering SourceForge, BasicQuery and Basic XSLT.

The original presentation can be found at:
http://www.blueslate.net/Presentations/

David Read
CTO
Blue Slate Solutions

June, 2006

Agenda

BLUE SLATE solutions

- **BasicQuery**
  - **Operation**
  - Source Code

BasicQuery

BLUE SLATE solutions

- BasicQuery is a Java-based application used to access databases through JDBC. It features a Swing-based GUI and includes capabilities useful to developers. It also produces timing information, which is valuable during tuning exercises.
- Latest Release Notes (version 01.02.00):
  - Production release encompassing support for 6 languages, soft-reference-based results cache and numerous options for controlling application behavior, including reporting of client and server details. Given the variety of improvements over the last production release, it is well worth your time upgrading. On the source code side, this version completes the integration of log4j, JUnit and Cobertura. The test suite is fairly complete, other than automating the testing of the GUI components.

basicquery.sourceforge.net

BLUE SLATE solutions

BasicQuery SourceForge Project Homepage - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://basicquery.sourceforge.net/

Customize Links Free Hotmail Windows Marketplace Windows Media Windows Yahoo! Mail Action Controller: Exc... Yahoo!

Blue Slate Solutions : Strate... Login to Scalix Web Access BasicQuery SourceForg... SourceForge.net: SourceFo... what is Trove categorization... Definition of trove - Merriam...

BasicQuery

SOURCEFORGE.net

Project Page | Application Installer

Latest News

Production Release (Version 01.02.00) is available for download as a file release 01.02.00, from our CVS repository (tagged v01-02-00), or as an InstallAnywhere® installer.

Reference

Feature List

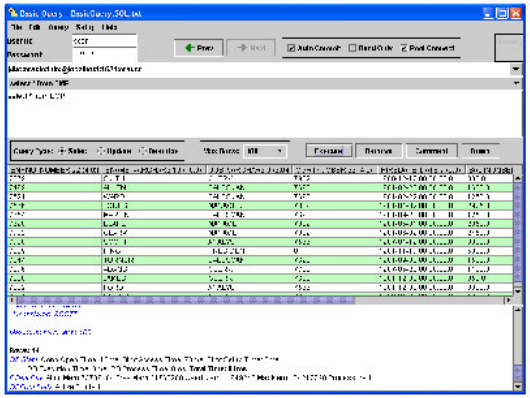
BasicQuery is a Java-based application used to access databases through JDBC. It features a Swing-based GUI and includes capabilities useful to developers when testing SQL statements against a database. It also produces timing information, which is valuable during tuning exercises. Development on BasicQuery began in early 2003 and it has been used on various projects since then.

We have leveraged BasicQuery in diverse situations which involved debugging, testing, and performance tuning. Key features of the application include easy access to each SQL statement entered; ability to save sets of SQL statements in separate files; support for parameterized statements; a provision for multiple return values; execution of sets of statements; storage of each connect URL used; and exporting of results, both data output and timing details, to a CSV file. Given its Java heritage, BasicQuery has been used successfully on multiple platforms.

BasicQuery can use any JDBC-compliant driver. The list of drivers to be loaded is stored in a parameter file. The driver packages themselves are loaded dynamically at startup from a user-configurable location. We regularly access MySQL, Oracle, and Sybase databases. The Apache Jakarta Commons Pool Library can be used, allowing timing tests to reflect the impact of pooled versus non-pooled connections.

On the source code side many changes continue to be made. With the 01.02.00 production release out, the next set of betas will be focused on the following feature set:

Priority	Functionality
HIGH	Allowing multiple SQL statements to be pasted/imported at once
HIGH	Storing SQL with embedded CRs (to allow stored proc definitions, etc)
HIGH	Control of Font Family and Font Size
MED	User Documentation
MED	XML support, bi-directional
MED	Reverse-engineering table-create scripts
LOW	Add GUI coverage to unit testing
LOW	Help System integration



BasicQuery's Main Screen -- Enlarged View

Done

Operational Highlights

- Support Provided for 6 Languages
- Report DB Server Information
- Caching of Previous Results
- Control of Result Row Display Coloring
- Memorized Queries can be Reordered
- SQL Statement and Corresponding Connection String can be Linked
- Clipboard Support is Provided for Copying Result Rows
- Stores Each Connection URL
- Stores Each SQL Statement
- Displays Result Set Data in a Table
- Provides Details of Each Statement Execution
- Optional Use of a Pooled Connection
- Control of Auto-Commit on the JDBC Connection
- Control of Read-Only Setting on the JDBC Connection
- Use Multiple SQL Statement Files
- Logging of SQL Execution Statistics to a CSV File
- Log All Select Query Result Data
- Write Current Select Results to CSV File
- Save BLOB Field Contents to a File
- Raw Export Mode
- Prevent Line Breaks Between Records
- Sort By Multiple Columns
- Create a Select Query Based on the Current Results
- Create an Insert Statement Based on the Current Results
- Create an Update Statement Based on the Current Results
- Create a Select * Query Based on the Selected Cell
- Obtain Metadata Based on Selected Cell
- Run All Queries as a Batch
- Create Parameterized SQL Statements
- View Metadata Based on a Select Statement
- Limit Rows Retrieved
- Remove a SQL Statement from the Set of Stored Statements
- Comment-out a SQL Statement in the Set of Stored Statements

Interface

Query: Run Delete Comment-Out Next in History

User Id/Password

Connect Strings

SQL History

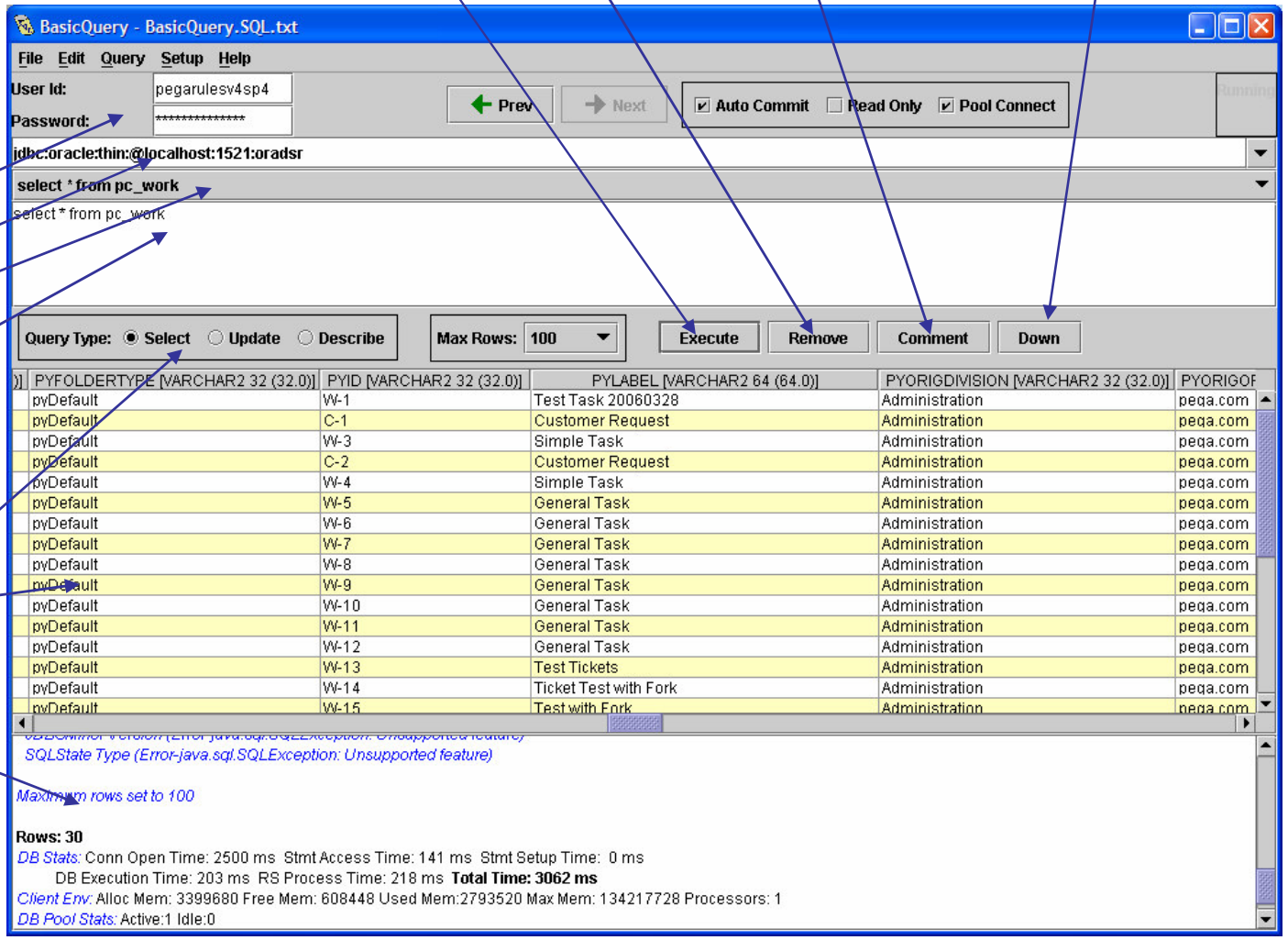
Current SQL

Row Limit

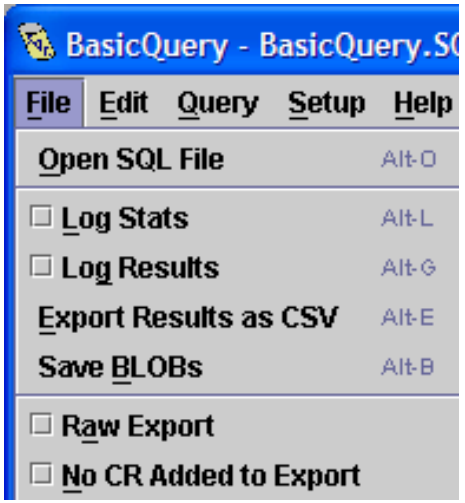
Statement Type

Results

Messages



File Menu



Open SQL File

Groups of SQL statements may be stored in separate files, this options loads or creates a file of SQL statements

Log Stats

Appends all execution statistics to a CSV file

Log Results

Appends all query results to a CSV file

Export Results as CSV

Writes current results to a CSV file

Save BLOBS

Writes any BLOB field(s) in the currently selected row to individual file(s)

Raw Export

Write embedded CR/LF to CSV file

No CR Added to Export

Do not place a CR at end of each record in CSV file

Query Menu

Select Statement

Using the current results, create a select statement where each column is separately named

Insert Statement

Using the current results, create an insert statement where each column is separately named

Update Statement

Using the current results, create an update statement where selected columns are used for a where clause and the remainder are in the set clause

Select *

Create a select * statement where table name is data in the selected cell

Describe Select *

Create a select * statement where table name is data in the selected cell and query type is set to “Describe”

Reorder Queries

Allow query history to be manually reordered

Run All Queries

Run each query in the query history a given number of times

Setup Menu

Language
- Override the system default language

Display DB Server Info
- Report DB server metadata in the messages area

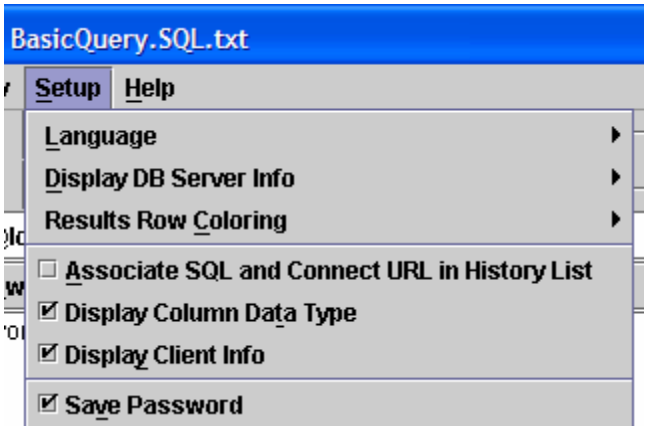
Results Row Coloring
- Apply an alternating color scheme to the result display

Associate SQL and Connect URL in History List
- Cause the connect URL to change based on which connection was last used for a given SQL statement

Display Column Data Type
- Include column metadata in the results column heading

Display Client Info
- Report client information in the messages area

Save Password
- Insecurely store the last password used



Parameter-Based SQL

- Sometimes you are calling stored procedures or functions that require IN and/or OUT parameters.
- BasicQuery supports a syntax for this purpose.
- IN Parameters
  - **$PARAM[IN,***DataType***,***YourData***]$**
  - e.g. **$PARAM[IN,String,Albany]$** creates an **IN** parameter of type **String** with the value **Albany**.
- OUT Parameters
  - **$PARAM[OUT,***DataType***]$**
  - e.g. **$PARAM[OUT,Integer]$** creates an **OUT** parameter of type **Integer**.

Example of Parameter-Based SQL

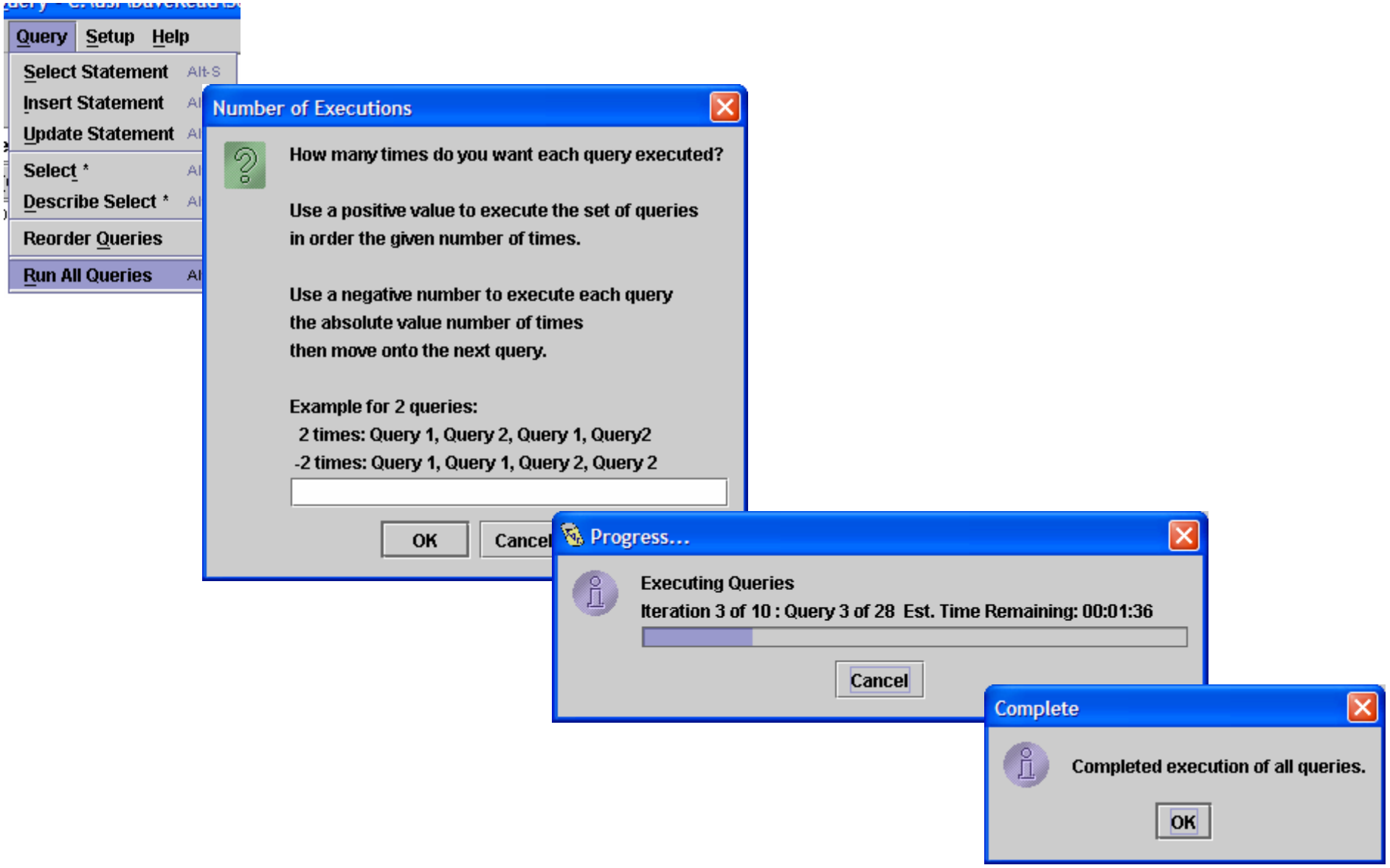
- Imagine a DB function that takes two IN parameters (integers) and returns two OUT parameters (integers - the sum and difference respectively). Further, the function itself returns an integer status.

```
CREATE OR REPLACE FUNCTION test_sp (
    num_1      IN      NUMBER,                    -- input number 1
    num_2      IN      NUMBER,                    -- input number 2
    the_sum    OUT     NUMBER,       -- return the sum
    the_diff   OUT     NUMBER                     -- return the diff
    ) RETURN NUMBER
```

- How do we use it from BasicQuery?
- {$PARAM[OUT,INTEGER]$ = call test_sp(8,-8, $PARAM[OUT,INTEGER]$,$PARAM[OUT,INTEGER]$)}
  - Note Oracle's JDBC driver wants the braces ({}) around the function call
- Where are the OUT parameter values displayed?
  In the message area:

Out Param (Integer 0): **-1**
Out Param (Integer 1): **0**
Out Param (Integer 2): **16**

Run Queries Repeatedly for Timings



Collected Statistics

Microsoft Excel - BasicQuery.Stats.xls

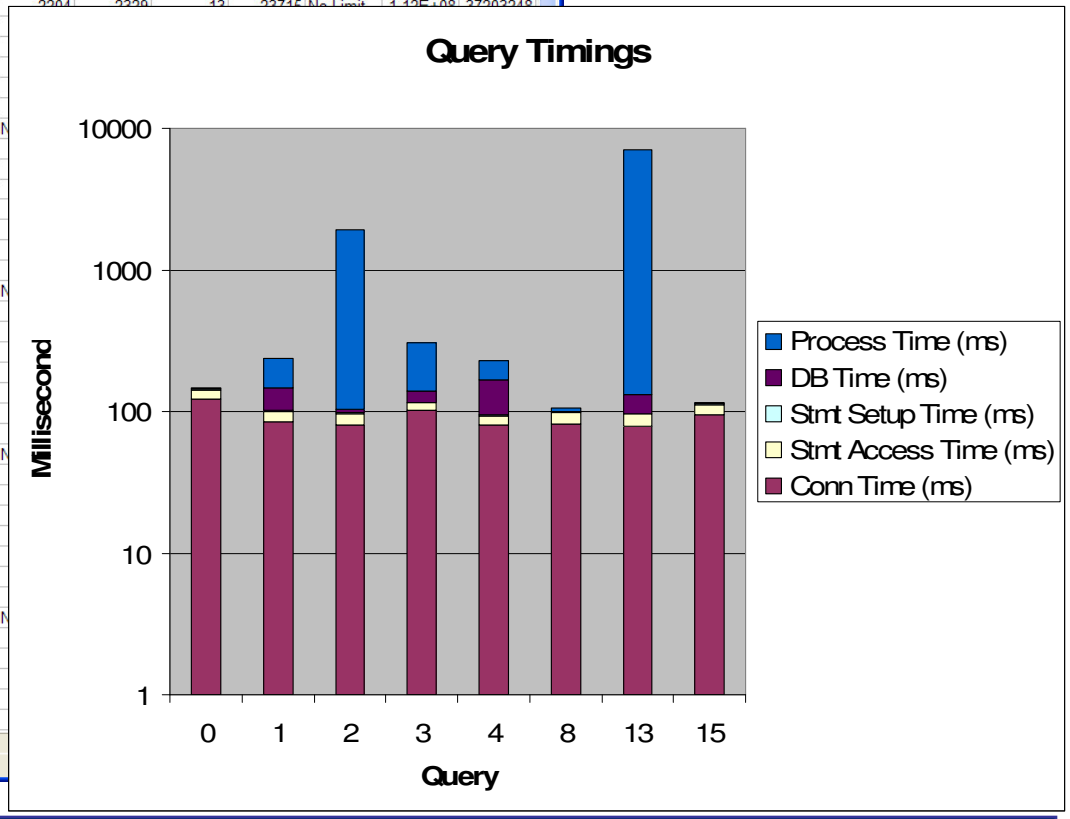
File Edit View Insert Format Tools Data Window Help

A1 Query Num

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Query Num	Query	Date	Conn Time	Stmt Acce	Stmt Setu	DB Time (r	Fetch Time	Process Ti	Total Time	Columns	Rows	Max Rows	Alloc Mem	Free Mem
2	0	{$PARAM[OUT,INTEGER]$ =	20:57.7	891	125	0	15	N/A	N/A	1031	0	-1	No Limit	66715648	61830440
3	1	select * from all_tables	20:58.1	94	47	0	62	N/A	172	375	46	71	No Limit	66715648	63023192
4	2	select * from all_objects	21:00.5	78	47	0	0	N/A	2032	2157	13	23715	No Limit	66715648	47183840
5	3	select * from all_objects wher	21:02.2	94	31	0	32	N/A	171	328	13	1109	No Limit	66715648	48379720
6	4	select * from user_source	21:02.5	93	32	0	62	N/A	78	265	4	2411	No Limit	66715648	47162952
7	8	select * from all_tables where	21:02.7	78	16	0	15	N/A	0	109	46	40	No Limit	66715648	43332648
8	13	select * from PR4_RULE	21:09.1	78	47	0	78	N/A	6188	6391	106	15485	No Limit	66715648	9975408
9	15	select * from all_objects wher	21:11.5	109	16	0	0	N/A	0	125	13	1	No Limit	66715648	9985424
10	0	{$PARAM[OUT,INTEGER]$ =	21:11.6	94	31	0	0	N/A	N/A	125	0	-1	No Limit	66715648	8525888
11	1	select * from all_tables	21:11.9	94	15	0	47	N/A	78	234	46	71	No Limit	66715648	10713432
12	2	select * from all_objects	21:14.2	79	31	0	15	N/A							
13	3	select * from all_objects wher	21:15.8	109	16	0	16	N/A							
14	4	select * from user_source	21:16.1	94	0	0	78	N/A							
15	8	select * from all_tables where	21:16.3	94	31	0	0	N/A							
16	13	select * from PR4_RULE	21:23.0	78	15	0	47	N/A							
17	15	select * from all_objects wher	21:25.5	94	15	0	0	N/A							
18	0	{$PARAM[OUT,INTEGER]$ =	21:25.6	78	0	0	0	N/A							
19	1	select * from all_tables	21:25.9	78	16	0	46	N/A							
20	2	select * from all_objects	21:27.9	78	31	0	0	N/A							
21	3	select * from all_objects wher	21:29.5	93	0	0	16	N/A							
22	4	select * from user_source	21:29.8	78	0	0	79	N/A							
23	8	select * from all_tables where	21:30.0	94	15	0	0	N/A							
24	13	select * from PR4_RULE	21:36.3	78	16	0	31	N/A							
25	15	select * from all_objects wher	21:38.8	94	15	0	0	N/A							
26	0	{$PARAM[OUT,INTEGER]$ =	21:38.9	78	16	0	0	N/A							
27	1	select * from all_tables	21:39.1	78	16	0	31	N/A							
28	2	select * from all_objects	21:41.1	78	15	0	16	N/A							
29	3	select * from all_objects wher	21:43.2	110	15	0	16	N/A							
30	4	select * from user_source	21:43.5	78	16	0	62	N/A							
31	8	select * from all_tables where	21:43.7	62	16	0	0	N/A							
32	13	select * from PR4_RULE	21:49.9	78	32	0	31	N/A							
33	15	select * from all_objects wher	21:52.3	94	15	0	0	N/A							
34	0	{$PARAM[OUT,INTEGER]$ =	21:52.4	78	16	0	0	N/A							
35	1	select * from all_tables	21:52.6	78	15	0	47	N/A							
36	2	select * from all_objects	21:54.5	78	16	0	0	N/A							
37	3	select * from all_objects wher	21:56.2	125	0	0	32	N/A							
38	4	select * from user_source	21:56.5	94	15	0	78	N/A							
39	8	select * from all_tables where	21:56.6	94	16	0	0	N/A							
40	13	select * from PR4_RULE	22:04.5	94	16	0	31	N/A							
41	15	select * from all_objects wher	22:06.9	125	16	0	0	N/A							
42	0	{$PARAM[OUT,INTEGER]$ =	22:07.0	78	15	0	0	N/A							
43	1	select * from all_tables	22:07.2	78	16	0	47	N/A							
44	2	select * from all_objects	22:09.1	78	16	0	0	N/A							
45	3	select * from all_objects wher	22:10.8	110	15	0	32	N/A							
46	4	select * from user_source	22:11.1	93	16	0	78	N/A							
47	8	select * from all_tables where	22:11.3	94	15	0	0	N/A							

BasicQuery.Stats

Ready



Agenda

BLUE SLATE solutions

- **BasicQuery**
  - Operation
  - **Source Code**

Source Code Highlights

BLUE SLATE solutions

- java.util.PropertyResourceBundle to support multiple languages
- Separate class loader for JDBC drivers
- Soft-reference based cache for holding results
- javax.swing.table.TableCellRenderer for rendering data in JTable
- Apache Commons Pooling
- Log4j, JUnit and Cobertura

java.util.PropertyResourceBundle

BLUE SLATE solutions

- Alternative to subclassing the ResourceBundle or ListResourceBundle classes.
  - Avoids hardcoding text in the ResourceBundle class
  - Avoids needing to create your own property file interface code to go between your proprietary property file and a ResourceBundle class
- Note that a PropertyResourceBundle is the last option checked by the environment, so a ResourceBundle subclass will trump the PropertyResourceBundle for the same language and locale.

Isolating the Resource Access

BLUE SLATE solutions

- BasicQuery uses a class, Resources, to provide access to the ResourceBundle-based information.
- Overridden getString() method takes the resource key and optionally 1 or more text arguments.
- getString() then uses java.text.MessageFormat to integrate the arguments into the text.
- The developer of the resource must provide parameter arguments ({0}, {1}, …) at the appropriate points in the text.
  - These arguments must agree with the expected arguments for the message as used in the source code.
- Example:

  Resources.getString("msgParamInDesc", type + "", data)
- English (BasicQueryResources_en.properties)

  msgParamInDesc=In Parameter: {0}:{1}

  In Parameter: 4:8
- French (BasicQueryResources_fr.properties)

  msgParamInDesc=Dans Le Paramètre : {0}:{1}

  Dans Le Paramètre : 4:8

java.lang.ClassLoader

BLUE SLATE solutions

- Abstract class for loading classes
  - Subclass it in order to create your approach to loading classes
- Each class holds a reference to the class loader that provided it
- Typically override **findClass()** when creating your own ClassLoader subclass.
- BasicQuery has a DynamicClassLoader class to deal with the JDBC driver classes
  - Issue is that the set of driver libraries is not known until runtime so they cannot be defined on the application's classpath
  - Actually leverages the java.net.URLClassLoader concrete class to reference the libraries found in a JDBC libraries directory (configured in BasicQuery's properties file).
  - The URLClassLoader is responsible for loading these classes while the DynamicClassLoader class overrides the **loadClass()** method to intercept the request for a class.
  - First the system class loader is checked, and if the class is not found the URLClassLoader instance is used.

Using the DynamicClassLoader Instance

BLUE SLATE solutions

```
DynamicClassLoader dbClassLoader;

dbClassLoader = new
  DynamicClassLoader(archives);

…

constructDriver =
  Class.forName(driverClass, true,
  dbClassLoader).getConstructor(null);

DriverManager.registerDriver(new
  DynamicDriver((Driver)constructDriver.
  newInstance(null)));
```

References

BLUE SLATE solutions

- From the java.lang.ref package documentation…
  - An object is ***strongly reachable*** if it can be reached by some thread without traversing any reference objects. A newly-created object is strongly reachable by the thread that created it.
  - An object is ***softly reachable*** if it is not strongly reachable but can be reached by traversing a soft reference.
  - An object is ***weakly reachable*** if it is neither strongly nor softly reachable but can be reached by traversing a weak reference. When the weak references to a weakly-reachable object are cleared, the object becomes eligible for finalization.
  - An object is ***phantom reachable*** if it is neither strongly, softly, nor weakly reachable, it has been finalized, and some phantom reference refers to it.
  - Finally, an object is ***unreachable***, and therefore eligible for reclamation, when it is not reachable in any of the above ways.

java.lang.ref.SoftReference

BLUE SLATE solutions

- Wrapper around object reference

```
Object obj = new Object();
SoftReference soft = new SoftReference(obj);
obj = null;
```

  - The Object instance is now softly reachable (no strong references exist).

```
obj = soft.get();  // Will return instance or null
```

- No control over when the reference will be cleared.
  - Guaranteed that GC cycle will have reclaimed all softly (or weaker) referenced objects before throwing an OutOfMemoryError.

Leverage Soft References

BLUE SLATE solutions

- Quick way to add a memory-sensitive cache
- Not much code to write in order to wrap and then retrieve instances
- Must always test returned instance in case null (instance was cleared).
- BasicQuery uses this in the QueryHistory class which holds one query in the history list as well as its results (wrapped in a soft reference)

```
public void setResults(TableModel pResults) {
  results = new SoftReference(pResults);
}
public TableModel getResults() {
  return (TableModel)results.get();
}
```

- When Retrieving the Cached Results

```
histModel = ((QueryHistory)historyQueries.
  get(histPosition)).getResults();
if (histModel != null) {
  sorter = new TableSorter(histModel);
  …
}
```

javax.swing.table.TableCellRenderer

BLUE SLATE solutions

- Interface defining the method:

```
getTableCellRendererComponent(JTable table,
 Object value, boolean isSelected, boolean
 hasFocus, int row, int column)
```

  - The method's job is to obtain the rendering component, place the value in it and return the instance
- Renderer's job is to represent value as appropriate. For instance if value is a number it may be represented as a color, dial position, digits, …
- Must design the renderer component to be very efficient, it is called for every visible cell on the table whenever the table is redrawn (invalidating window sections, scrolling, …)

ColoredCellRenderer

BLUE SLATE solutions

- BasicQuery combines the TableCellRenderer and the actual rendering component in one class.
  - Extends JLabel and implements TableCellRenderer
- The underlying implementation actually uses the same renderer instance for all cells
  - Very efficient – but beware side effects
  - Don't hold any cell data in your rendering component unless you think very carefully about how you associate it with the cell.
- ColoredCellRenderer does hold data related to the foreground and background color to be used, but that is determined by row of each call.

getTableCellRendererComponent()

BLUE SLATE solutions

```
public Component getTableCellRendererComponent(JTable
  table, Object value,
      boolean isSelected,
      boolean hasFocus, int row,
      int column) {

    if (value != null) {
      setText(value.toString());
    } else {
      setText("");
    }

    setColor(table, isSelected, hasFocus, row, column);

    return this;
  }
```

Apache Commons Pooling

BLUE SLATE solutions

- Generic Pooling: org.apache.commons.pool
- DB-centric extensions: org.apache.commons.dbcp
- Allow pool to be defined in XML file or created dynamically.
  - Currently BasicQuery creates the pool dynamically based on connection URL and id/password provided.
- Allows performance testing timings to be applicable to systems where a connection pool is being used

Dynamically Creating the DB Connection Pool

BLUE SLATE solutions

```
private void setupDBPool(String connectURI, String userId, String password) throws
  SQLException {
   GenericObjectPool connectionPool = new GenericObjectPool(null);
   configurePool(connectionPool, connectURI, userId, password);
   PoolingDriver driver = new PoolingDriver();
   driver.registerPool(DBPOOL_NAME, connectionPool);
}
private void configurePool(GenericObjectPool connPool, String connectURI, String
  userId, String password) throws Exception {
   String lowerCaseConnectURI, validationQuery;
   lowerCaseConnectURI = connectURI.toLowerCase(); validationQuery = null;
   if (lowerCaseConnectURI.startsWith("jdbc:sybase")) {
     validationQuery = "select getdate()";
   } else if … { … }
   connPool.setWhenExhaustedAction(GenericObjectPool.WHEN_EXHAUSTED_BLOCK);
   connPool.setMaxWait(5000); connectionPool.setMaxIdle(1);
   DriverManagerConnectionFactory connectionFactory =
       new DriverManagerConnectionFactory(connectURI, userId, password);
   PoolableConnectionFactory poolableConnectionFactory = new
       PoolableConnectionFactory(connectionFactory, connPool, null, null,
       false, true);
   if (validationQuery != null) {
     connPool.setTestOnBorrow(true); connPool.setTestWhileIdle(true);
     poolableConnectionFactory.setValidationQuery(validationQuery);
   }
 }
```

Obtaining a Connection from the Pool

BLUE SLATE solutions

```
Connection conn;
conn = DriverManager.getConnection(
  "jdbc:apache:commons:dbcp:" +
  DBPOOL_NAME);
```

Log4J, JUnit, Cobertura

- Well documented tools
- Other CDJDN presentations cover JUnit and Cobertura