PQL Deep Dive Lab

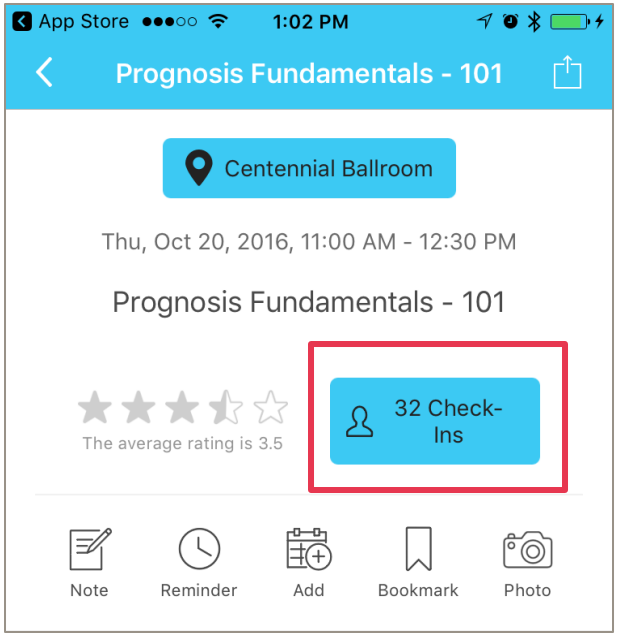
All about PQL

Wednesday, October 18, 2017

Shoaib Dilawar

Welcome!

Please check-in on the mobile app - *see your class record, remember what tests to take, and help us improve*



Summit prognosis

Logistics

- **WiFi SSID:** IRsummit
- **WiFi PW:** Prognosis
- Check the **sticker** on your badge for credentials to log in to your **Lab instance**

Agenda

- Introduction to PQL
- Troubleshooting PQL
- Retrieving Prognosis Data
- Using PQL Views
- Exercises

Learning Objectives

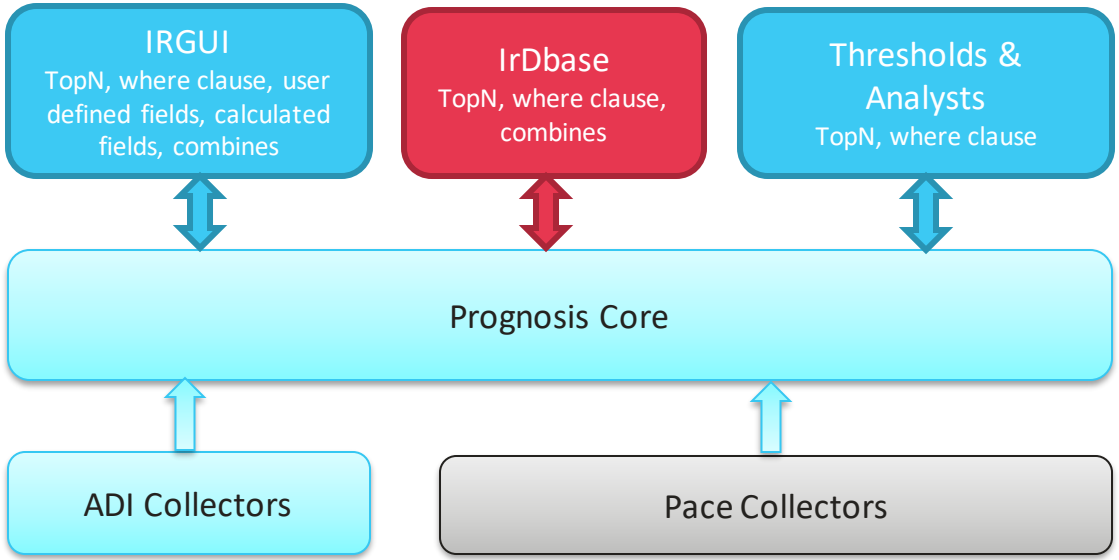
- After completing this course you should:
  - Be able to get filtered data from prognosis
  - Be able to write and run PQL queries
  - Be able to use advance calculations in PQL
  - Understand PQL views
  - Troubleshoot common PQL issues

Introduction to PQL

- PQL – Prognosis Query Language
- PQL is a subset of SQL
  with a number of semantic and syntax extensions to fit the Prognosis data request model.
- PQL is easy to understand
  remains fairly understandable to anyone with basic SQL knowledge.

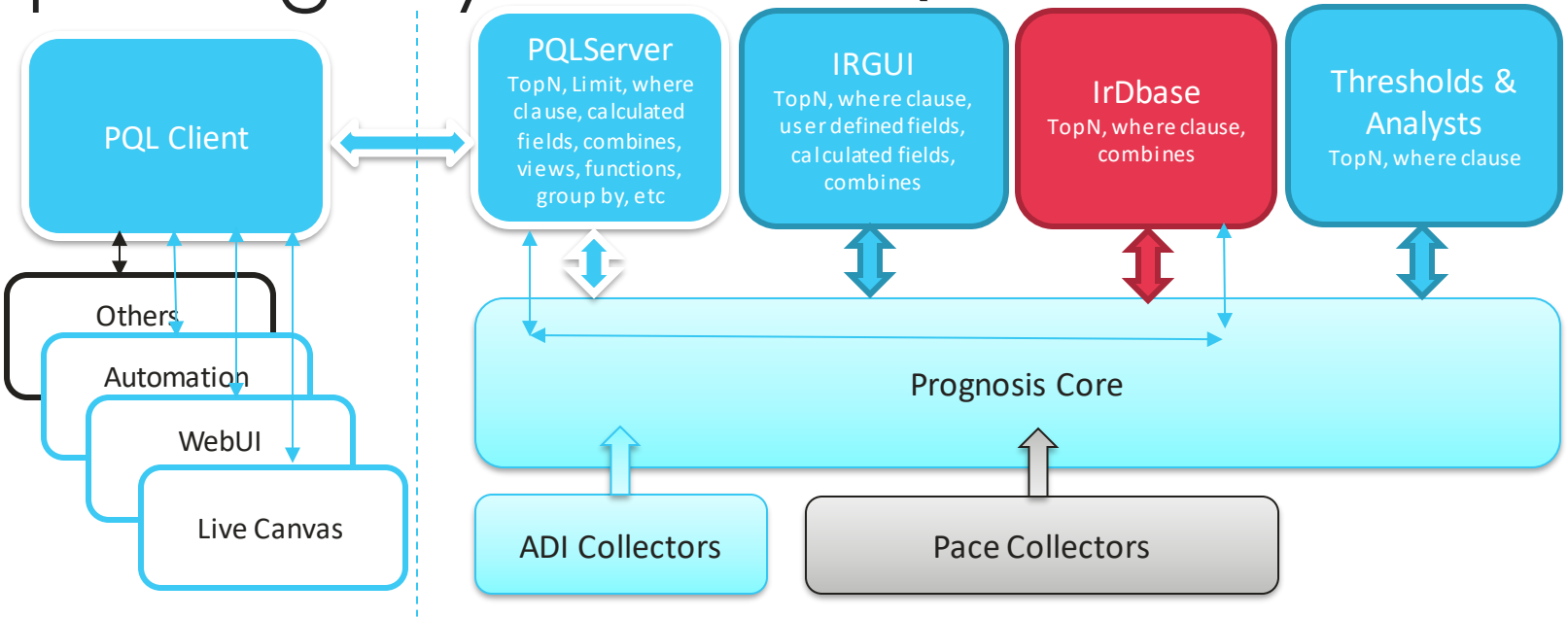
Introduction to PQL

Requesting Layer before PQL



Introduction to PQL

Requesting Layer After PQL



Troubleshooting PQL

Using PQL Command Line Utility

irpqlcli.exe is located in Prognosis Server folder and can be used to execute PQL queries.

The best way to execute it is to go to Prognosis Server\Configuration folder and execute the command:

..\irpqlcli "SELECT * FROM NtCpu" –r

Type ..\irpqlcli without arguments to see a short documentation on how to use it.

Interesting options

Option	Description
-r	Display output in a readable format.
-t	Test option. Statements are only compiled but not executed.
-a	Analyze. Outputs the time needed to execute each internal step.
-b param[=value]	Specify parameter values. Use one for each param.

Retrieving Prognosis Data

- Like SQL, PQL uses SELECT statement to retrieve data:

```
SELECT * FROM NTProcess
```

- Retrieving a single field from a record

```
SELECT ProcessName FROM NTProcess
```

- Retrieving multiple fields from a record

```
SELECT ProcessName, ProcessCreationTime,
BusyPercent FROM NTProcess
```

Retrieving Prognosis Data

- Specifying a collection interval

```
SELECT ProcessName From NTProcess EVERY 1 SECOND
```

  Valid EVERY time specifiers: second, seconds, minute, minutes, hour, hours, day, and days

- The default is 10 seconds if no EVERY clause is specified

```
EVERY 1 is same as EVERY 1 SECOND or EVERY 1 SECONDS
```

- This is an extension to the SQL standard

Retrieving Prognosis Data

- Specifying Prognosis Nodes

```
SELECT Meta.Node, ProcessName FROM NTProcess
NODE ‘\ir-pbxmon-srv’
```

- Specifying multiple nodes

```
SELECT Meta.Node, ProcessName FROM NTProcess
NODE ‘\ir-pbxmon-src’, ‘\dev-ricardom7’
```

Retrieving Prognosis Data

- Specifying ALL Nodes

```
SELECT Meta.Node, ProcessName FROM
NTProcess NODE ALL
```

- Specifying CURRENT Node

```
SELECT Meta.Node, ProcessName FROM NTProcess
NODE CURRENT
```

Retrieving Prognosis Data

Prognosis Meta Fields

Meta Field	Description
Meta.Node	Name of the Prognosis Node where data was collected from. This can be an appliance (fake) node, or a real machine where Prognosis is running.
Meta.Interval	Start time of interval when the data was collected
Meta.Associate	Value of the associate for the record being collected
Meta.ServerTime	For queries that request data from: • multiple sources (e.g. multiple nodes and/or associates) • or the current node • or a virtual appliance monitored by the current node The value is the same as Meta.Interval. Otherwise the value is the time when PQL server received data from Prognosis, rounded to the nearest wall-clock aligned interval boundary.

Retrieving Prognosis Data

Scalar Query

returns exactly one row containing one or more columns

```
SELECT Now() AS Today

SELECT Count(*) AS ProcessCount FROM NtProcess

SELECT TOP 1 ProcessName, PrivateMB FROM NtProcess
ORDER BY BusyPercent Desc
```

Retrieving Prognosis Data

- Filtering data using Where Clause
  Where clauses are expressions evaluated on every row of data. If the expression evaluates to TRUE the row is included in the result. Otherwise if the expression evaluates to FALSE the row is excluded.
- Where clause operations
  - Comparison operators =, <>, <, <=, >, >=, IS NULL, IS NOT NULL
  - Arithmetic operators +, -, *, /, %, MOD
  - Logical operators AND, OR, NOT, IN, NOT IN
  - Expression operators (,)
  - Wild card filtering with CONTAINS, MATCHES and MATCHES REGEX
  - Build-in Functions

Retrieving Prognosis Data

- Using IN and NOT IN

  The **IN** operator is used to help reduce the need for multiple OR conditions in a SELECT statement. It returns true if the expression to the left of the IN operator is the same as one of the items in the list.

```
SELECT ProcessId, ProcessName FROM NtProcess WHERE
ProcessName IN {'irpqlsrv', 'iradicol', 'irnetrtr'}
```

- NOT IN operator

  The **NOT IN** operator returns true if the expression to the left of the NOT IN operator is not equal to any of the items in the list.

```
SELECT ProcessId, ProcessName FROM NtProcess WHERE
ProcessName NOT IN {'irpqlsrv', 'iradicol', 'irnetrtr'}
```

Retrieving Prognosis Data

Using MATCHES

MATCHES operator allow text pattern matching operations using wildcard characters:

Char	Description
*	Matches zero or more characters
?	Matches exactly one character

SELECT ProcessName FROM NtProcess WHERE ProcessName MATCHES 'ir*col*'

NOT MATCHES inverts the results and return all processes which do not match the pattern.

MATCHES and NOT MATCHES are not case sensitive.

Retrieving Prognosis Data

- Using MATCHES REGEX
  The Matches Regex and Not Matches Regex operators allow regular expression pattern matching operations.

```
SELECT ProcessId, ProcessName FOM NtProcess WHERE
ProcessName MATCHES REGEX '^ir(.*)srv'
```

- NOT MATCHES REGEX
  Produces the reverse result of MATCHES REGEX

```
SELECT ProcessId, ProcessName FOM NtProcess WHERE
ProcessName NOT MATCHES REGEX '^ir(.*)srv'
```

Retrieving Prognosis Data

- ## Understanding Calculated Fields

  Data returned by Prognosis records is often not available in a format needed by an application:

  - Text in different fields need to be concatenated
  - Calculations need to be applied to numeric values
  - Values of one data type need to be converted to another data type
  - Data in a column needs to be summarized

  Like SQL, PQL calculated fields are the result of expressions passed on the field list of a SELECT statement:

  SELECT field list FROM data source WHERE filtering conditions

Retrieving Prognosis Data

- Concatenating text

  SELECT ExecutablePath + '\' + ProcessName + '.exe' AS ProcessPath FROM NtProcess WHERE ExecutablePath IS NOT NULL

- Numerical calculations

  SELECT CpuNumber, (100 – BusyPercent) AS "Idle Percent" FROM NtCPU

- Data Type conversion and summarization

  SELECT CAST(count(*) AS STRING) + ' Running Processes' FROM NtProcess

Retrieving Prognosis Data

PQL Built-in Functions

Built-in functions can be used in calculated fields and where clause expressions. There are 7 categories:

Category	Description
Display functions	Transform data into an appropriate display format
Numeric functions	Commonly used mathematical functions
Meta Data functions	Return meta information, such as HostName, PrognosisVersion etc.
Data Type functions	Operate on Prognosis specific data such as response time etc.
Text handling functions	Provide commonly operations on text data
Timestamp functions	Provide commonly operations on dates and times
Miscellaneous functions	Other miscellaneous functions

Retrieving Prognosis Data

Interesting Built-in Functions

```
SELECT Meta.Node, HostName(), PrognosisVersion() FROM NtProcess GROUP BY Meta.Node

SELECT SubString(ProcessName, 3) FROM NtProcess WHERE ProcessName MATCHES 'ir' GROUP BY ProcessName

SELECT now()

SELECT DatePart(Year, Now()) AS "Year", DatePart(Month, Now()) AS "Month", DatePart(Day, Now()) AS "Date"

SELECT SetTime(2016, 3, 17, 7, 15, 39600)

SELECT 'Year ' + Cast(DatePart(Year, Now()) AS STRING)
```

Retrieving Prognosis Data

Using CASE expressions

A CASE expression allows you to specify complex, multi-level conditions in your SELECT statement fields list and where clause. CASE expressions have two modes: test various values against a single computed expression, or as a sequence of conditional statements.

CASE expression can be used in Where Clauses

Retrieving Prognosis Data

CASE using single computed value

```
SELECT ProcessName,
CASE ProcessName
     WHEN 'irdbase'  THEN 'Prognosis Database'
     WHEN 'irnetrtr' THEN 'Prognosis Net Router'
     WHEN 'irpqlsrv' THEN 'Prognosis PQL Server'
     ELSE 'Other Prognosis Process'
END AS Explanation
FROM NtProcess WHERE ProcessName MATCHES ‘ir*’
```

Retrieving Prognosis Data

CASE testing various values

```
SELECT ProcessName,
CASE
   WHEN ProcessName ='irdbase' THEN 'Prognosis Database'
   WHEN ProcessName = 'irnetrtr' THEN 'Prognosis Net Router'
   WHEN ProcessName ='irpqlsrv' THEN 'Prognosis PQL Server'
   WHEN ProcessName MATCHES ‘ir*’ THEN ‘Other Prognosis
Process’
   ELSE 'Other System Process'
END AS Explanation
FROM NtProcess GROUP BY ProcessName ORDER BY ProcessName
```

Retrieving Prognosis Data

Understanding Parameters

Parameters provide a simple and convenient way to create dynamic PQL queries.

When parameters are used in queries, the values used in the query is provided only at execution time.

This is specially valuable for PQL Views (discussed later in this presentation), that can be created with parameters and the actual values can be delayed until the view is started.

A parameter is simply the character @ followed by a name. Example: @param1, @Node, @cluster.

Retrieving Prognosis Data

Using Parameters

SELECT @param

SELECT ProcessName, ProcessId FROM NtProcess
WHERE ProcessName = @name NODE @node

SELECT CASE @param WHEN '' THEN 'Empty string'
ELSE 'Not empty string' END

Retrieving Prognosis Data

Where to use Parameters

- Calculated fields expressions
- Where clause expressions
- Associate clause
- Node clause
- Param clause
- Credentials clause
- Argument of any Execute command
- Arguments of PQL functions
  Except datepart argument of DatePart, DateAdd and DateDiff and the datatype part of Cast function

Retrieving Prognosis Data

Sorting Retrieved Data

By default PQL returns data as it receives it from Prognosis collectors. PQL provides the ORDER BY clause to enable the sorting of data returned.

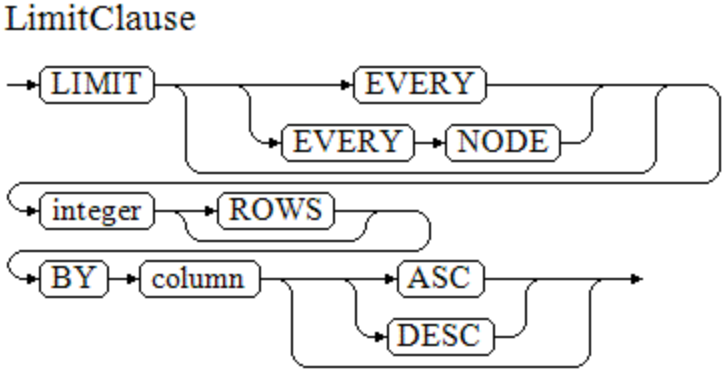
SELECT ProcessName AS Proc FROM NtProcess GROUP BY Proc ORDER BY Proc DESC

SELECT ProcessName FROM NtProcess WHERE ProcessName MATCHES 'ir' ORDER BY ProcessName ASC

SELECT ProcessName, ProcessId FROM NtProcess ORDER BY ProcessId

Retrieving Prognosis Data

- Limiting Data
  PQL has two clauses to limit the number of rows it produces: TOP and LIMIT
- TOP n
  limits the number of rows returned by PQL.
- LIMIT
  Prognosis passes this limit to Prognosis collectors to limit the number of rows of data they return.



Retrieving Prognosis Data

- TOP n example

```
SELECT TOP 1 Meta.Node, ProcessName, PrivateMB FROM
NtProcess NODE ALL

SELECT TOP 1 Meta.Node, ProcessName, PrivateMB FROM
NtProcess ORDER BY PrivateMB DESC NODE ALL
```

- LIMIT examples

```
SELECT Meta.Node, ProcessName, PrivateMB FROM NtProcess
LIMIT 1 BY PrivateMB DESC NODE ALL

SELECT Meta.Node, ProcessName, PrivateMB FROM NtProcess
    LIMIT EVERY 1 BY PrivateMB DESC NODE ALL
```

Retrieving Prognosis Data

UNION

- PQL can combine data from more than one data source (record or view)
- First query defines columns selected.
- Subsequent queries must provide value for each column.

```
SELECT 'Average CPU Busy' as Metrics, AVG(BusyPercent) as Value FROM NTCPU UNION SELECT 'Total Processes', sum(BusyPercent) FROM NTPROCESS WHERE ProcessName != 'Idle'
```

Summit prognosis

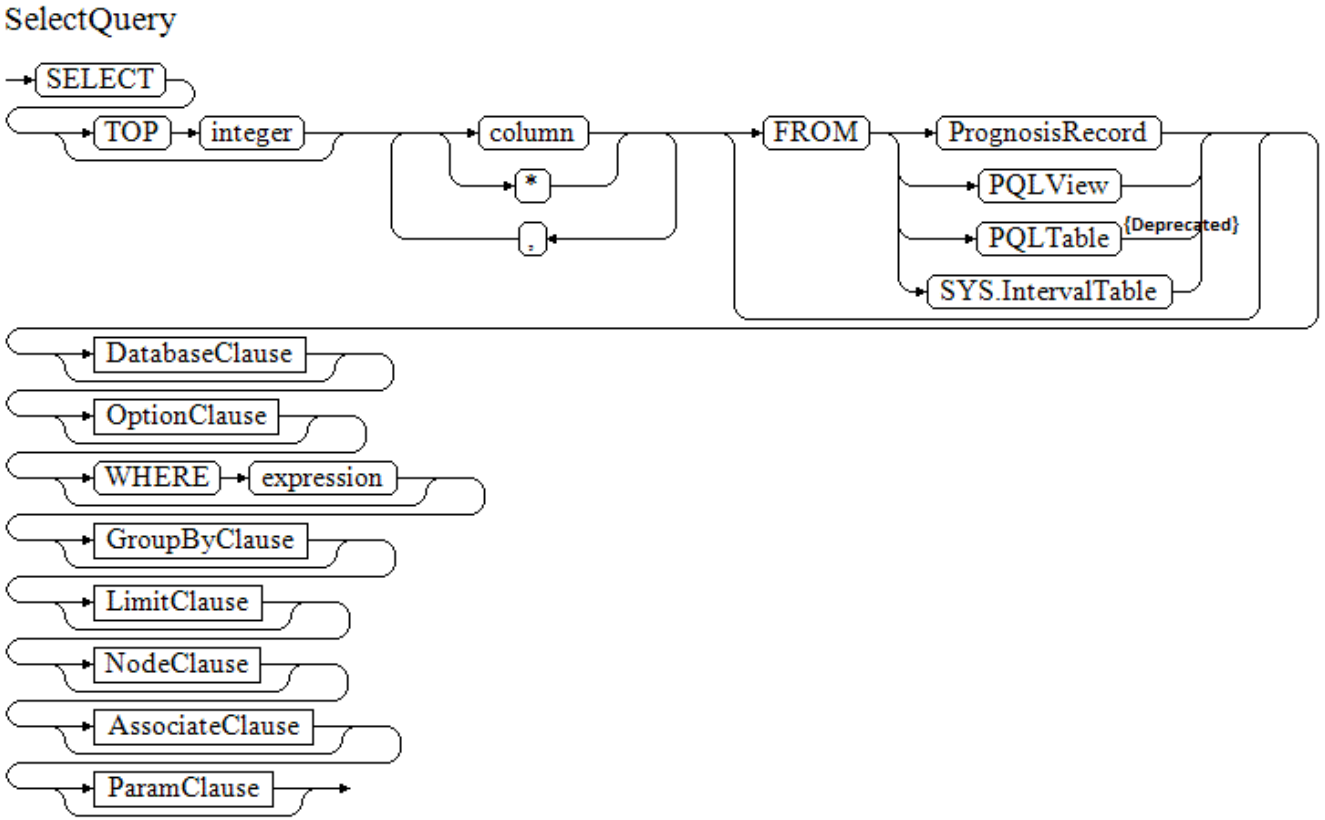
Retrieving Prognosis Data

Historical Data

- PQL can query historical data from a prognosis record
- DATABASE keyword is used to specify what database to use.
- DBNODE specifies where DB is running.
- PERIOD <from> TO <to> is used to specify the window within the database to look for data.

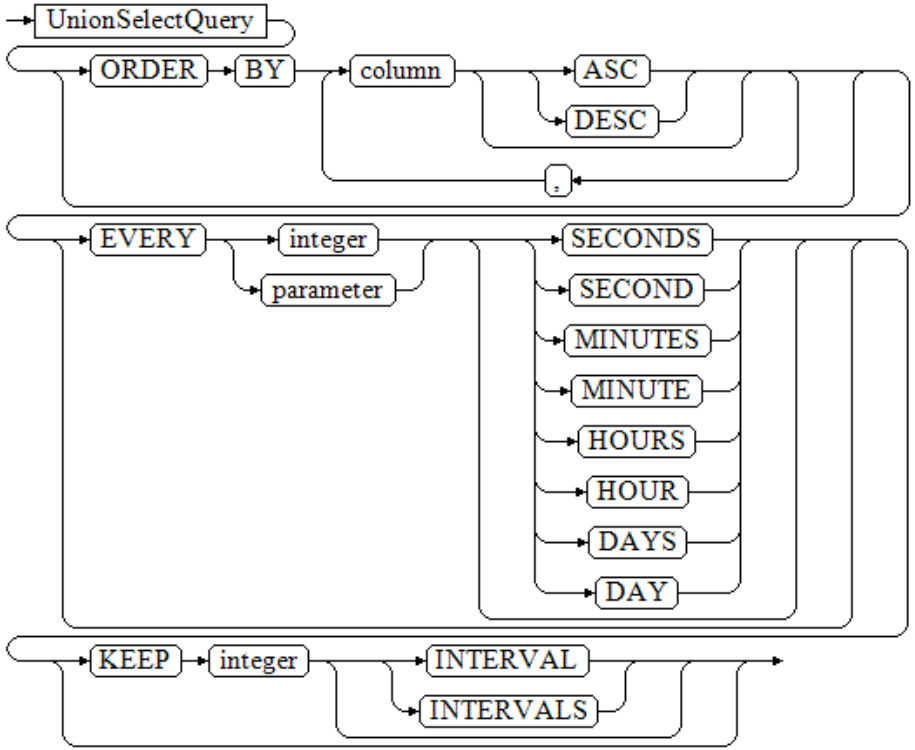
```
SELECT TOP 10 * FROM CMACalls DATABASE @rptDatabaseName
DBNODE @rptDatabaseNode PERIOD @rptFrom TO @rptTo
```

PQL Query Syntax



PQL Query Syntax

Select



Using PQL Views

- Understanding PQL Views
  So far all PQL SELECT statements we have seen will start a Prognosis Data Collection and retrieve data real time.
- Creating a PQL View will:
  - Save named queries for future use
  - Create a named query that retrieves data from a data source but presents a different column list
  - Run a named query in the background to
    - Keep a cache of historical intervals
    - Return data immediately

Using PQL Views

Creating Views

Views are created with the CREATE VIEW command:

CREATE VIEW <view name> AS <select statement>
[TIMEOUT n [SECONDS | MINUTES | HOURS] | TIMEOUT PERSISTENT]

TIMEOUT n [SECONDS | MINUTES | HOURS]

The view is not persistent and will stop requesting data after n units of time has passed without data being requested. The default time out is 3 intervals.

Using PQL Views

TIMEOUT PERSISTENT LOCAL

- Indicates the view will be stored in PQL configuration and will be restarted every time PQL is restarted.
- Data is kept locally, so needs more memory resources.
- Each new @prompt value would create a new instance of the view.
- Each role in RBS would create a new instance per role.

Using PQL Views

TIMEOUT PERSISTENT DISTRIBUTED

- View is persistent and the cache of data is distributed to all Prognosis nodes.
- Memory consumption is distributed across all nodes.
- Strict rules on qualification for distributed view.

Exercise

- Select all ir processes running on the monitoring node
- Select all ir processes running that are actually busy
- Show the accumulative busy percent for all ir processes

Exercise

- Show busy percent of all processes excluding ‘Idle’, refreshing every second, 10 times.
- Show 5 busiest processes excluding ‘Idle’
- Show details of a process named ‘irpqlsrv’ matching ProcessName to a prompt @ProcessName

Exercise

- Categorize processes that start with ir as ‘Prognosis Process’, Idle as ‘Not a Process’ and everything else as ‘Not a Prognosis Process’, showing Name, Category and BusyPercent.
- Show totals for above 3 categories.

Exercise

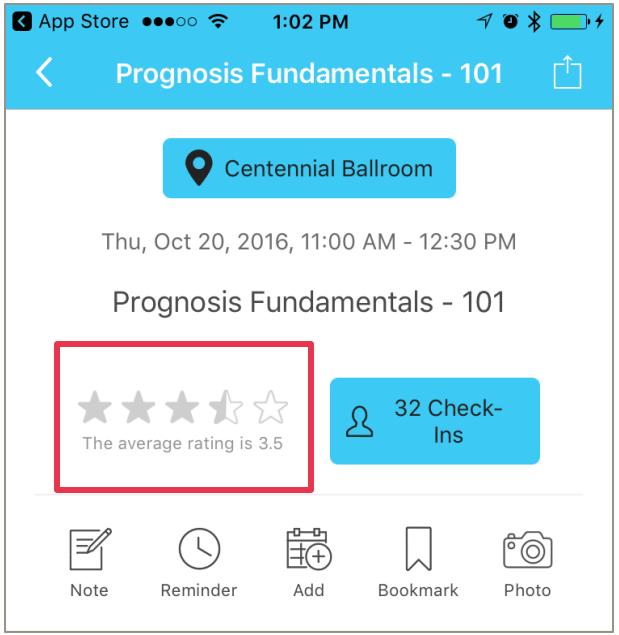
- Create a Persistent view for NtProcess refreshing every 30 seconds.
- Run the Category query against above view, is it any quicker?

Recap

- We've covered:
  - What were the most important points
  - What can they do now
  - What do they know
  - Where can they get more help (Online.Prognosis.com)
  - Your contact info (if you want)

Next Steps

- **Please Rate the Class**
- **Take the Knowledge Reinforcement Test**
- **Log On to Online.Prognosis.com** to download slides & ask questions
- *Every class rating gets you a chance to win prizes!*



Questions?