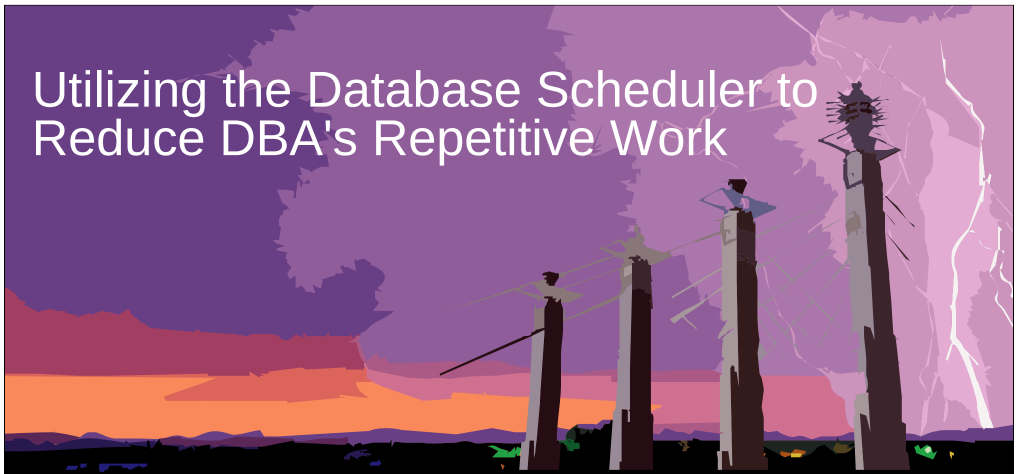


# Utilizing the Database Scheduler to Reduce DBA's Repetitive Work

John F. Miller III
IBM

A09
Tuesday, April 29, 2008 • 1:00 p.m. – 2:00 p.m.

2008 IIUG **Informix** Conference

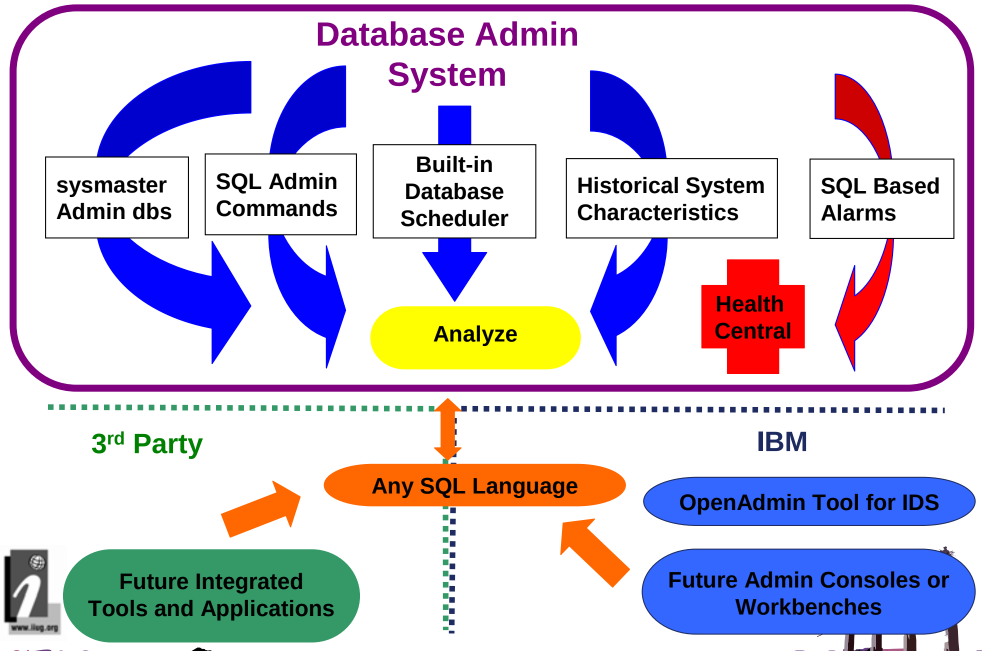
www.iiug.org

## Talk Overview

- What's the Benefit of the Database Scheduler ?
- What are Tasks and Sensors?
- Table Layouts
- Scheduler Threads
- How to use it

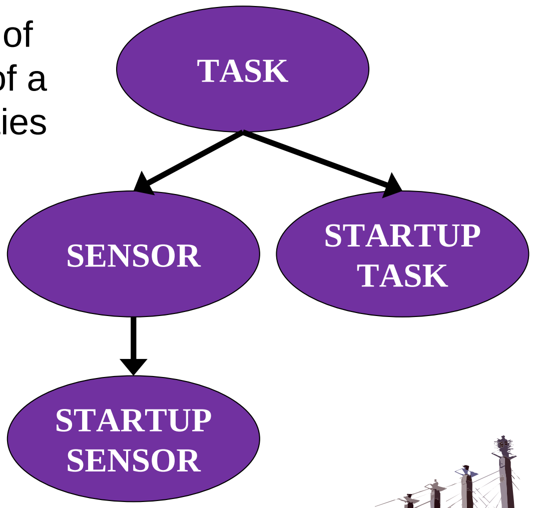
## Abilities the Database Scheduler Provides

- Execute an SQL operation or function in the database server based on a schedule, interval or event
- Periodically monitoring and collecting data from the database server into database tables for later analysis
  - Performance monitoring (Checkpoints, throughput, I/O)
  - Utilization monitoring (logs, disk space, memory)


Database Admin System
sysmaster Admin dbs
SQL Admin Commands
Built-in Database Scheduler
Historical System Characteristics
SQL Based Alarms
Analyze
Health Central
3rd Party
IBM
Any SQL Language
OpenAdmin Tool for IDS
Future Integrated Tools and Applications
Future Admin Consoles or Workbenches

## What are the Different Types of Tasks

- Task is the basic unit of work, all other types of a task inherit all properties of a task
- Four Basic Tasks
  - Task
  - Startup Task
  - Sensor
  - Startup Sensor

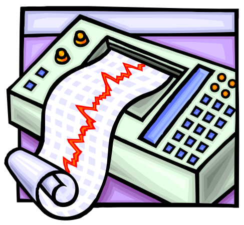


## What is a Task?

- Execute a specific action at a specific time or frequency
- No intention to evaluate eventually returned data
- The action can consists of
  - A single or set of SQL statements separated by a semicolon
  - A single or a set of  stored procedures separated by a semicolon also in combination with other SQL statements
  - A C or Java based UDR
- STARTUP TASK
  - Similar to a TASK but the specific action will be executed at database server startup time once

## What is a Sensor?

- Executes an action at a specified time and/or frequency with the intention to collect data about an object and store the data in a table
- The action of a sensor can consist of
  - One or multiple statements (most likely inserts) separated by a semicolon
  - One or multiple Stored Procedures, C or JAVA UDRs
- Provides a mechanism to purge expired data
- Provides the ability to create the storage table if it does not exist
- STARTUP SENSOR
  - Similar to a SENSOR, but the action will be executed only once at server startup time



## Actions Executed by Tasks & Sensors

- Any single SQL statement
  - Example of forcing a checkpoint

```
execute function task ( “onmode”,”c”)
```

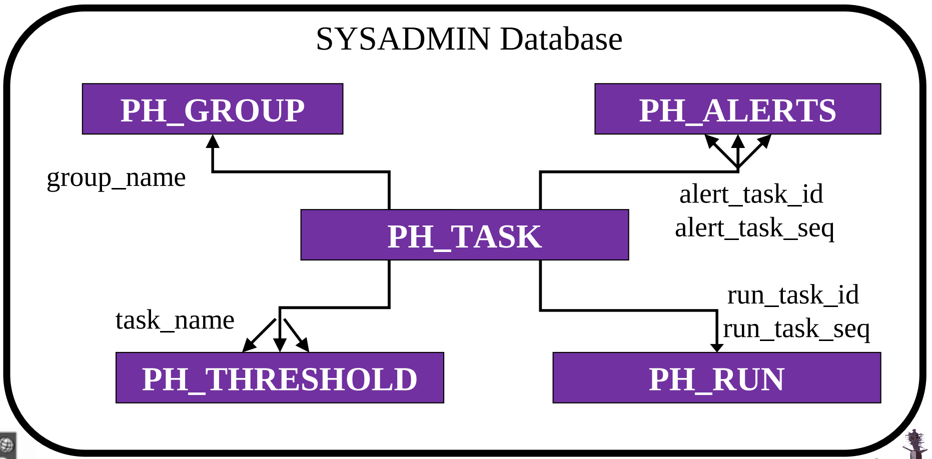
- Any compound SQL statement
  - Example of truncating a tables and deleting all order more than 1 day away

```
truncate table mytab; delete from orders where odate < today
```

- Any User Defined Routine (C, Java, SPL)
  - Execute the function *myfunction*

```
myfunction
```

## Database Scheduler Tables



## Table sysadmin:ph_task

- Each row in the table is a single task to be executed
- These columns describe the task and what is to be executed
- Items in Yellow are system supplied and maintained

| Column Name | Column Description |
|---|---|
| tk_id | The task id, ( a serial ) |
| tk_name | The task name, must be unique |
| tk_description | User supplied description of task |
| tk_group | The group this task is associated with (see ph_group) |
| tk_type | Task type (TASK, SENSOR, … ) |
| tk_dbs | The logging database current when executing the statement |
| tk_execute | The SQL statement or procedure to execute |
| tk_total_execution | The number of times this task has been executed |
| tk_total_time | The amount of time spent executing this task |
| tk_sequence | The latest sequence ID |
| tk_enable | True the task will be scheduled. |

## Table sysadmin:ph_task *(continued)*

- The following columns describe when the task is to execute
- Items in Yellow are system supplied and maintained

| **Column Name** | Description |
|---|---|
| tk_frequency | The interval the task should run |
| tk_start_time | Do not start task before this time (NULL = disabled) |
| tk_stop_time | After this time, the task does not execute(NULL = disabled) |
| tk_Monday | True if the task is to be execute on Monday |
| tk_Tuesday | True if the task is to be execute on Tuesday |
| tk_Wednesday | True if the task is to be execute on Wednesday |
| tk_Thursday | True if the task is to be execute on Thursday |
| tk_Friday | True if the task is to be execute on Friday |
| tk_Saturday | True if the task is to be execute on Saturday |
| tk_Sunday | True if the task is to be execute on Sunday |
| tk_next_execution | System calculated time of the next execution of the task |

## Table sysadmin:ph_task Sensor Requirements

- The following columns are used only be sensors, ignored by tasks
- A sensor result_table must have a column called “ID”. This will hold the task sequence id.

| **Column Name** | Description |
|---|---|
| tk_create | A “create table” statement executed before the task is run the first time. This table will store the data collected by the sensor |
| tk_result_table | The name of the table which stores the sensors data |
| tk_delete | • The interval after which the data will be purged<br>• Table must have a column called “ID” which holds the task sequence id |

## Special Values with SQL statements

- To use the task id or task sequence in an SQL statement
  - In an SQL statement in the tk_execute column uses the $DATA_TASK_ID will replace with the current task id from the ph_task table
  - In an SQL statement in the tk_execute column use the $ $DATA_SEQ_ID will be replaced with the current sequence from the ph_task table

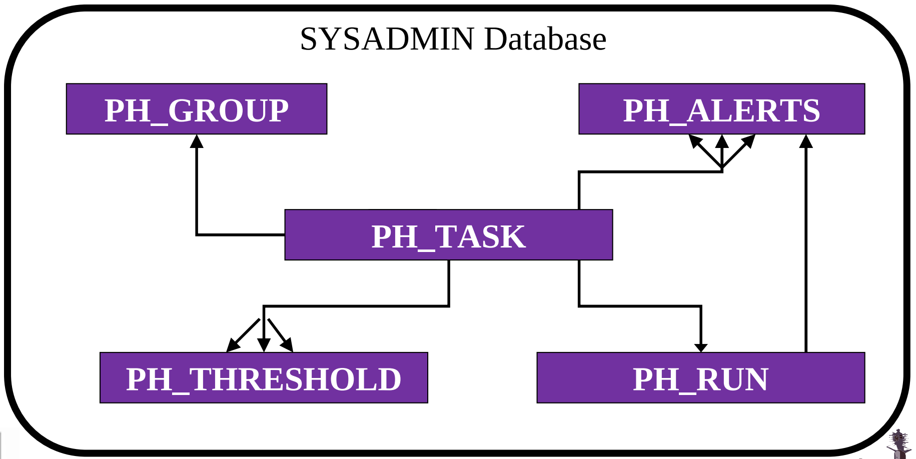
```
INSERT INTO tab (ID,c2) VALUES
( $DATA_SEQ_ID, $DATA_TASK_ID  )
```

## Special Values with Stored Procedures

- To use the task_id or task_sequence in stored procedure
  - Create a function which take two integers as parameters
  - Set tk_execute column in ph_task to the name of the function
  - The first argument is the task_id, the second argument is the task sequence

```
tk_execute = “my_function”
```

## Database Scheduler Tables



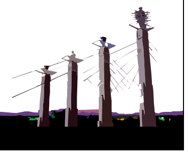
## Table sysadmin:ph_run

- One row will be created for each task executed
- Contains all information about the execution status of a specific task, such as:

| Column Name | Column Description |
|---|---|
| run_id | The run id, ( a serial ) |
| run_task_id | The task id associated with this run extery |
| run_task_seq | The unique invocation number of this task |
| run_retcode | The task's return code |
| run_time | The date and time of the tasks execution |
| run_duration | How long this task took in seconds |
| run_ztime | The last time onstat –z was executed |
| run_btime | The time the sever was booted |
| run_mttime | The current mt_counter |

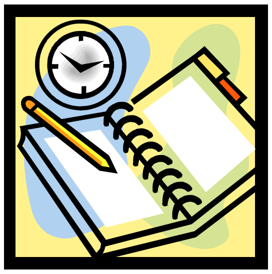
## Sysadmin Tables

- Sysadmin:ph_alert table
  - Contains user defined alerts which can be monitored by application programs, such as, OpenAdmin.
  - The table can be maintained by the system defined or user defined threads.
- Sysadmin:ph_threshold table
  - A list of thresholds or configuration values used by tasks
- Sysadmin:ph_group table
  - Task in the ph_task table are assigned to a group, this table contain a list of allowable group names



## Schedule Tasks using Criteria

- Execution window is between start time and stop time
- Runs with a configurable frequency
- Executes on specific days of week
- Any combination of the above
- Schedule to only run once at server startup time



## Database Scheduler Threads

- There are 3 persistent threads for the database scheduler
- A dbScheduler thread
  - Evaluates all tasks and schedules execution
- Two dbWorker threads
  - Threads are named dbWorker1 and dbWorker2
  - They perform the actual task

## View the Database Scheduler

onstat –g dbc

- Top section shows individual worker threads
  - Current Task
  - How much work this worker has done

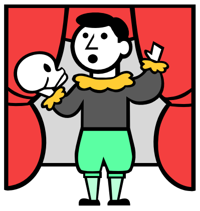
- The bottom shows the current schedule

```
Worker Thread(0)    4647efb0
=====================================
Task:                    464f8c20
Task Name:               mon_checkpoint
Task ID:                 7
Task Type:               SENSOR

WORKER PROFILE
    Total Jobs Executed               2
    Sensors Executed                  2
    Tasks Executed                    0
    Purge Requests                    2
    Rows Purged                       0

Scheduler Thread    46343e40
=====================================
Run Queue
    Empty
Run Queue Size        0
Next Task             8
Next Task Waittime    3482
```

## Built in Sensor & Tasks

| Sensor Name | Description |
|---|---|
| mon_command_history | Purges the command history table |
| mon_config | Saves any difference in the onconfig file |
| mon_config_startup | Save the onconfig file on every server startup |
| mon_profile | Save the server profile information |
| mon_vps | Collects the virtual processor timings |
| mon_checkpoint | Save information about checkpoints |
| mon_table_profile | Save table profile information |
| mon_table_names | Save the table names along with their create time |
| mon_users | Save profile information about each user |
| check_backup | Check to ensure backups have been done |
| Alert Cleanup | Purges the ph_alert table |
| Auto Update Statistics Refresh | Execute the Update Statistics commands |
| Auto Update Statistics Evaluation | Determine which table need to be updated and develop the Update Statistics commands |

## Examples



## Create a New Scheduler Group

- Allow for grouping of similar functionality
- OAT uses to keep similar results together

```
INSERT INTO ph_group
VALUES
(0,"EXAMPLES","Example and Demos")
```

## Simple Administration Task

- Task executes a checkpoint every 2 minutes, between the hours of 8AM and 7PM on Monday, Wednesday and Friday.

```
INSERT INTO ph_task
( tk_name,
tk_description,
tk_type,
tk_group,
tk_execute,
tk_start_time,
tk_stop_time,
tk_frequency,
tk_Monday,
tk_Tuesday,
tk_Wednesday,
tk_Thursday,
tk_Friday,
tk_Saturday,
tk_Sunday)
```

```
VALUES
( "Example Checkpoint",
"Example to do a checkpoint every 2 minutes.",
"TASK",
"EXAMPLES",
"EXECUTE FUNCTION admin('checkpoint')",
DATETIME(08:00:00) HOUR TO SECOND,
DATETIME(19:00:00) HOUR TO SECOND,
INTERVAL ( 2 ) MINUTE TO MINUTE,
't',
‘f’,
't',
‘f’,
't',
‘f’,
‘f’);
```

## Simple Administration Task

- Insert a row in to a table ex1_tab every minute between 8AM and 5 PM every day of the week.

```
INSERT INTO ph_task(
tk_name,    tk_description,
tk_type,    tk_group,
tk_execute,
tk_start_time,
tk_stop_time,
tk_frequency
) VALUES (
"Example One",
"Insert a row.",
"TASK",      "EXAMPLES",
"INSERT INTO ex1_tab(c1) VALUES (1)",
DATETIME(08:00:00) HOUR TO SECOND,
DATETIME(17:00:00) HOUR TO SECOND,
INTERVAL ( 1 ) MINUTE TO MINUTE) )
```

## Simple Sensor

- Creates the table, if it does not exist
- Executes the insert statement every minute between 8AM and 5 PM
- Deletes any data older than 10 minutes

```
INSERT INTO ph_task(
tk_name,        tk_description,
tk_type,        tk_group,     tk_result_table,
tk_create,      tk_execute,
tk_start_time, tk_stop_time,
tk_frequency,   tk_delete
) VALUES (
"Example Two", "Insert into ex2_tab every
minute, keeping 10 data points.",
"SENSOR",   "EXAMPLES", "ex2_tab",
"create table ex2_tab(ID integer, c2 integer)",
"insert into ex2_tab(ID,c2) values(
$DATA_SEQ_ID, $DATA_TASK_ID  )",
DATETIME(08:00:00) HOUR TO SECOND,
DATETIME(17:00:00) HOUR TO SECOND,
INTERVAL ( 1 ) MINUTE TO MINUTE),
INTERVAL ( 10) MINUTE TO MINUTE) )
```

## Advanced Sensor

- Creates the table, if it does not exist
- Executes the insert statement every minute between 8AM and 5 PM
- Deletes any data older than 1 hour

```
INSERT INTO ph_task(
tk_name,        tk_description,
tk_type,        tk_group,     tk_result_table,
tk_create,      tk_execute,
tk_start_time,  tk_stop_time,
tk_frequency,   tk_delete
) VALUES (
"Example Profile", "This will collect an onstat
-p every minute and keep the data for 1 hour.",
"SENSOR",  "EXAMPLES",  "ex_profile",
"CREATE TABLE ex_profile(ID INTEGER, name
VARCHAR(20), c2 BIGINT)",
"INSERT INTO ex_profile SELECT $DATA_SEQ_ID,
name, value FROM sysmaster:sysprofile",
DATETIME(08:00:00) HOUR TO SECOND,
DATETIME(17:00:00) HOUR TO SECOND,
INTERVAL ( 1 ) MINUTE TO MINUTE,
INTERVAL ( 1 ) HOUR TO HOUR )
```

## Creating a Task with Dynamic Parameters

- Add the parameter to the *ph_threshold* table
- Linked to task by task_name column

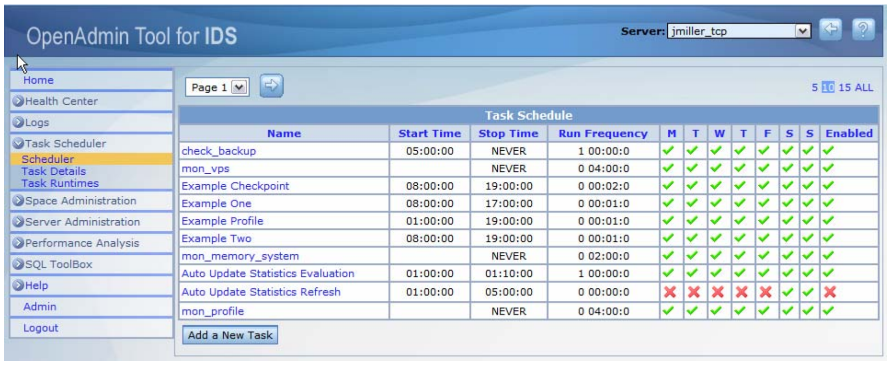
```
INSERT INTO ph_threshold
(id,
name,
task_name,
value,
description)
VALUES
 (0,
 "EXAMPLE RETENTION",
 "Example Delete with Param",
 "00 0:15:00",
 "Any values in column c2 older than
this interval will be purged.");
```

## Creating a Task with Dynamic Parameters

- Task executes between 5AM and midnight every 5 minutes and deletes any data from ex1_tab which is older than a "*interval*"
- The interval is retrieved from the *ph_threshold* table

```
INSERT INTO ph_task
( tk_name, tk_description, tk_type, tk_group, tk_execute,
tk_start_time, tk_stop_time, tk_frequency)  VALUES
( "Example Delete with Param",
"This example shows you how to utilize parameters",
 "TASK", "TABLES",
 "DELETE FROM ex1_tab where c2 < (
      SELECT MAX(current - value::INTERVAL DAY to SECOND)
      FROM ph_threshold
 WHERE name = 'EXAMPLE RETENTION' ) ",
 DATETIME(00:05:00) HOUR TO SECOND,
 NULL, INTERVAL ( 5 ) MINUTE TO MINUTE );
```

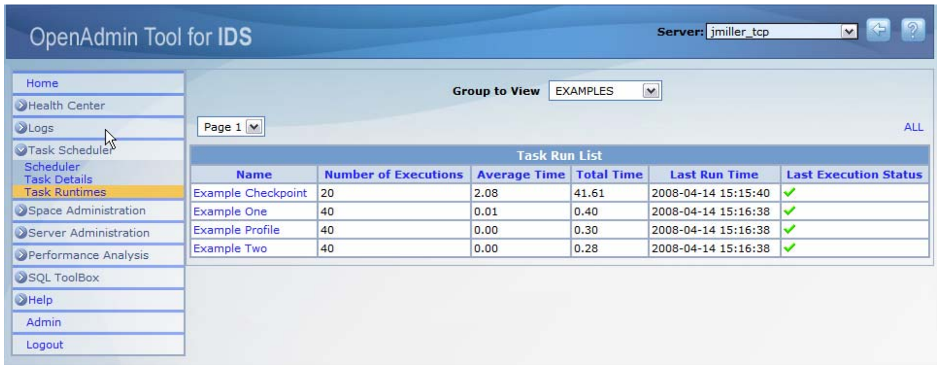
## Viewing Task Schedules from OAT



Task Schedule

| Name | Start Time | Stop Time | Run Frequency | M | T | W | T | F | S | S | Enabled |
|---|---|---|---|---|---|---|---|---|---|---|---|
| check_backup | 05:00:00 | NEVER | 1 00:00:0 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| mon_vps | | NEVER | 0 04:00:0 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Example Checkpoint | 08:00:00 | 19:00:00 | 0 00:02:0 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Example One | 08:00:00 | 17:00:00 | 0 00:01:0 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Example Profile | 01:00:00 | 19:00:00 | 0 00:01:0 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Example Two | 08:00:00 | 19:00:00 | 0 00:01:0 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| mon_memory_system | | NEVER | 0 02:00:0 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Auto Update Statistics Evaluation | 01:00:00 | 01:10:00 | 1 00:00:0 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Auto Update Statistics Refresh | 01:00:00 | 05:00:00 | 0 00:00:0 | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ |
| mon_profile | | NEVER | 0 04:00:0 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

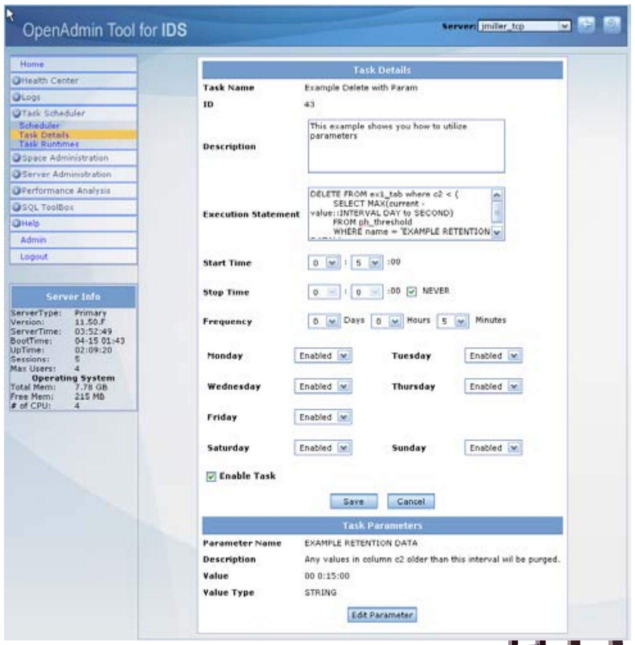
## Viewing Tasks Run Times from OAT



Task Run List

| Name | Number of Executions | Average Time | Total Time | Last Run Time | Last Execution Status |
|---|---|---|---|---|---|
| Example Checkpoint | 20 | 2.08 | 41.61 | 2008-04-14 15:15:40 | ✓ |
| Example One | 40 | 0.01 | 0.40 | 2008-04-14 15:16:38 | ✓ |
| Example Profile | 40 | 0.00 | 0.30 | 2008-04-14 15:16:38 | ✓ |
| Example Two | 40 | 0.00 | 0.28 | 2008-04-14 15:16:38 | ✓ |

## Viewing an Individual Tasks Schedule with Parameters



Server: jmiller_tcp

**Server Info**

|  |  |
|---|---|
| ServerType: | Primary |
| Version: | 11.50.F |
| ServerTime: | 03:52:49 |
| BootTime: | 04-15 01:43 |
| UpTime: | 02:09:20 |
| Sessions: | 5 |
| Max Users: | 4 |

**Operating System**

|  |  |
|---|---|
| Total Mem: | 7.78 GB |
| Free Mem: | 215 MB |
| # of CPU: | 4 |

**Task Details**

|  |  |
|---|---|
| **Task Name** | Example Delete with Param |
| **ID** | 43 |
| **Description** | This example shows you how to utilize parameters |

**Execution Statement**

```
DELETE FROM ex1_tab where c2 < (
        SELECT MAX(current -
value::INTERVAL DAY to SECOND)
        FROM ph_threshold
        WHERE name = 'EXAMPLE RETENTION
```

|  |  |
|---|---|
| **Start Time** | 0 : 5 :00 |
| **Stop Time** | 0 : 0 :00 ✓ NEVER |
| **Frequency** | 0 Days 0 Hours 5 Minutes |

|  |  |  |  |
|---|---|---|---|
| **Monday** | Enabled | **Tuesday** | Enabled |
| **Wednesday** | Enabled | **Thursday** | Enabled |
| **Friday** | Enabled |  |  |
| **Saturday** | Enabled | **Sunday** | Enabled |

✓ **Enable Task**

**Task Parameters**

|  |  |
|---|---|
| **Parameter Name** | EXAMPLE RETENTION DATA |
| **Description** | Any values in column c2 older than this interval wil be purged. |
| **Value** | 00 0:15:00 |
| **Value Type** | STRING |

www.iiug.org