Sensor & Task functions in IDS 11

David Jay
IBM Corporation

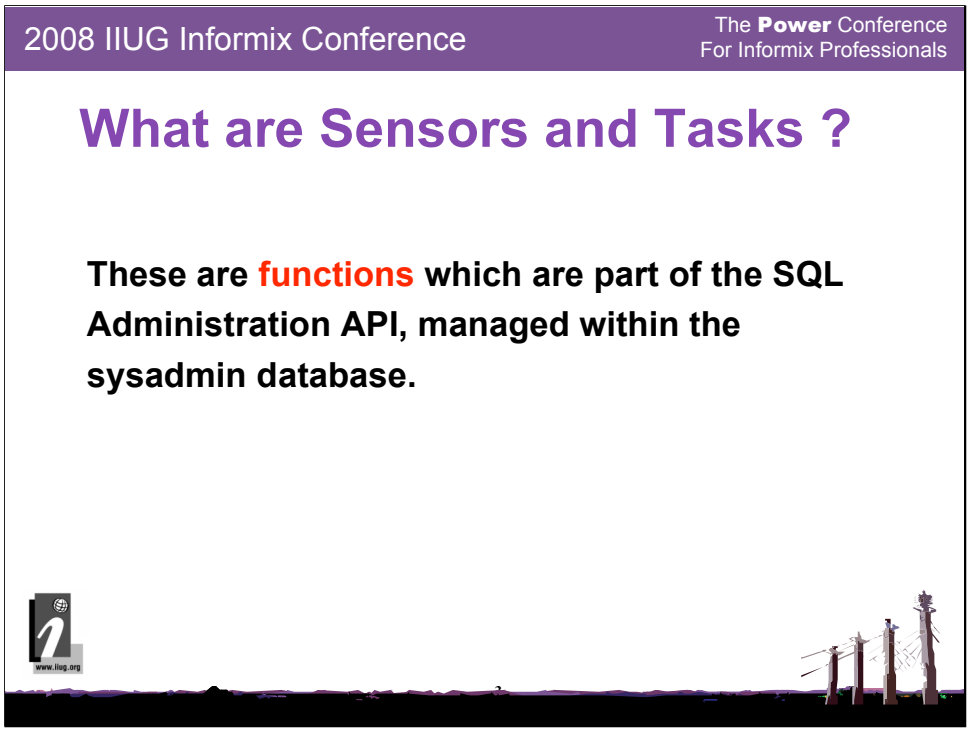
Session D12
Tuesday, April 29, 2008 • 4:40 – 5:40 p.m.

2008 IIUG Informix Conference

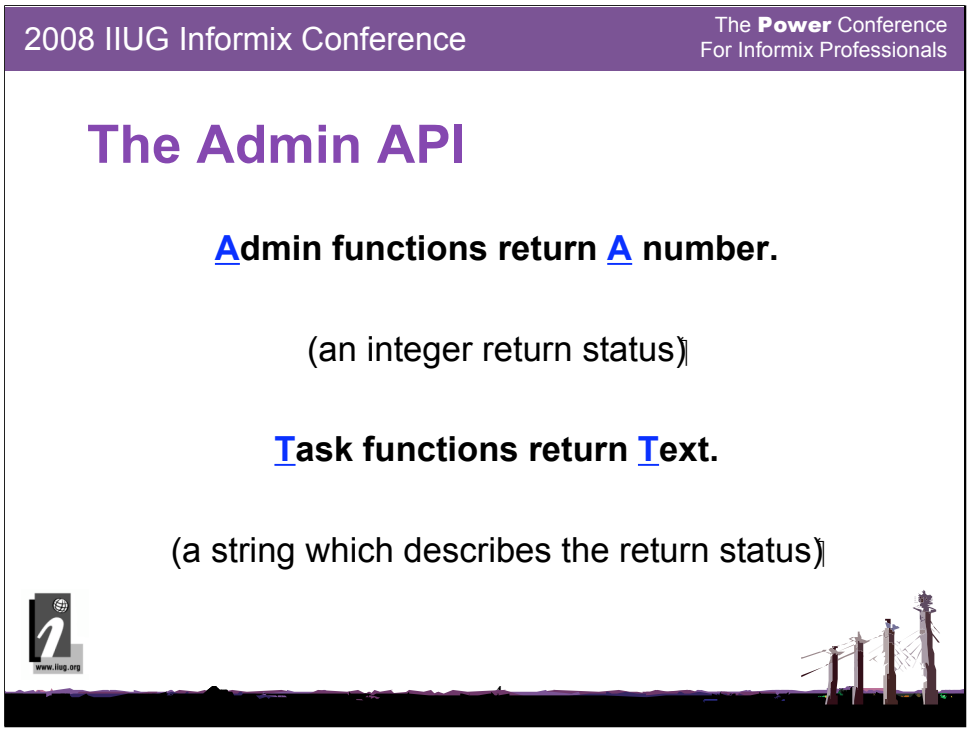
www.iiug.org

Topics Covered

- What are Sensors and Tasks in IDS 11?
- How they are instantiated, stored and used
- Common Usage
  - Productivity/Performance
- How to build a sensor
- Q&A
- Resources



It is also known as the Admin API, or Scheduler API, since there is a scheduler process that manages the whole thing.



Both types of functions perform the command and also write a result to the command _history table.

If the function succeeds but the write to the command_history table fails, it will write a message to the online.log

(It is 2 separate transactions)

What do Sensors and Tasks do?

- **Help the dba to administer one *or more* IDS 11 instances using a system based on SQL commands**
- **Minimize complexity and syntax errors**
- **These task functions can**
  - **Help manage resources**
  - **Allow on-the-fly configuration changes**
  - **Handle scheduling and running of routines**
  - **Perform system validation**
  - **Help to avoid complex shell scripts and command lines**

Tasks can automate scheduling or can be set to trigger on a condition.

Tasks allow you to store results for executed commands in order to track trends.

What is a Task?

Tasks are functions used to implement an administrative job or routine. When the task is run, at a minimum, it will write a completion status message to the command_history table.

What is a Sensor?

A Sensor is a task which not only implements a routine and records a command _history reference, it can also *collect and save* information.

3 Main parts of a Task:

- A definition, with command parameters which are stored as a row in the ph_task table
- A command function which you run
- A returned integer and lvarchar, providing a result for the command

If the task is also a sensor...

The task definition in ph_task table will include:

- **A SQL object, such as a stored procedure or function, to *collect* the information**
- **A table definition to create the table that will *hold* results for what you collect**
- **Interval information to determine *how and when* the sensor needs *to run, and how often to delete* data that has been stored for a while**

Creating a Task or Sensor

We need to recognize –

- **the elements that form the command syntax**
- **the defining elements in the PH_TASK table**

Examples of the command syntax:

task ('command');

task ('command', 'argument')

task ('command', size);

task ('command', 'argument', size);

The Command syntax

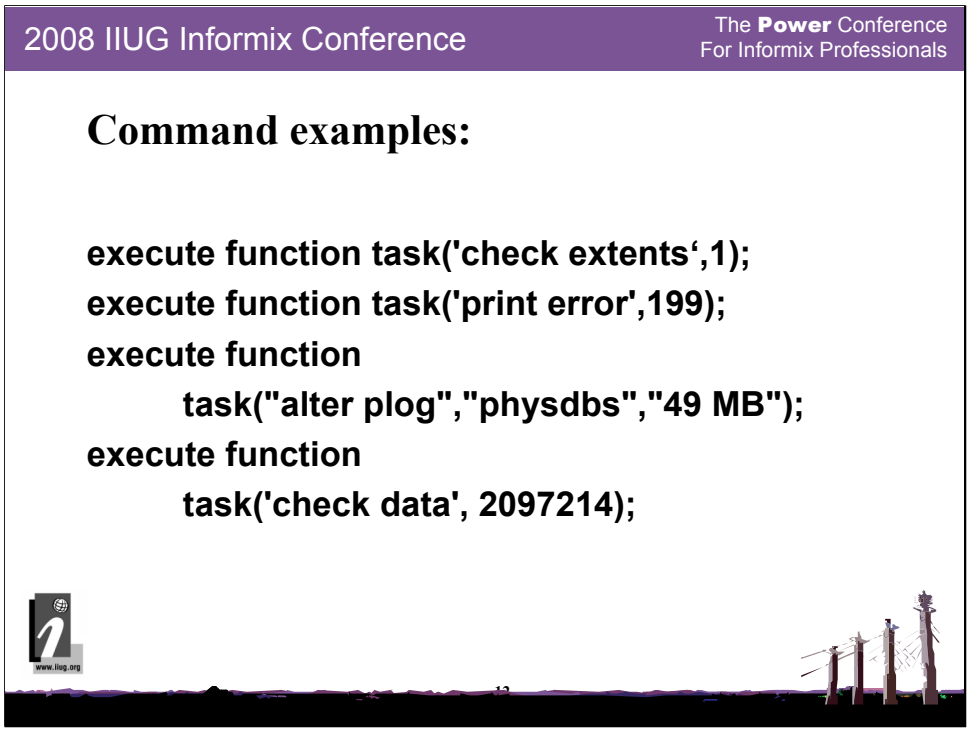
task ('command');

task ('command', 'argument')

task ('command', size);

task ('command', 'argument', size);

- **size is needed if the command specifies an offset, storage object, or buffer.**
- **size is a number followed by the unit abbreviation, with no space in between number and unit. (unit options include KB,B,MBGB,TB,PB). Default is KB when not specified.**



Check extents,1 == checks extents in the first dbspace.

“Check extents” == with no parameters – checks all the dbspaces.

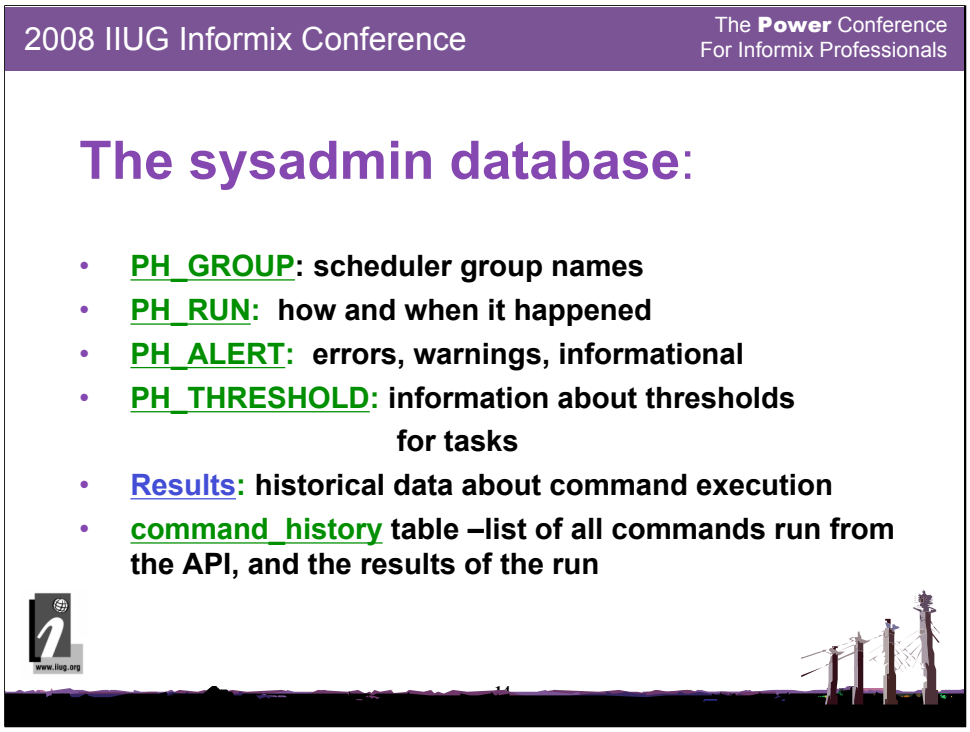
Check data can only be done with a partnum.

The task and the admin functions are created through a routine called db_install.sql, which is visible under $INFORMIXDIR/etc/sysadmin/.

If you look at the function definitions in db_install.sql, you will discover that task functions can handle up to 10 parameters.

Usage Notes:

- **These commands can be executed to remote instances, allowing a DBA to administer multiple instances through one command window.**
- **The SQL API has more than 80 command variations already.**



Collectively, the sysadmin database contains the tables used by the Scheduler:

Using task or admin commands the administrator can execute functions and procedures or schedule them to run at predefined times.

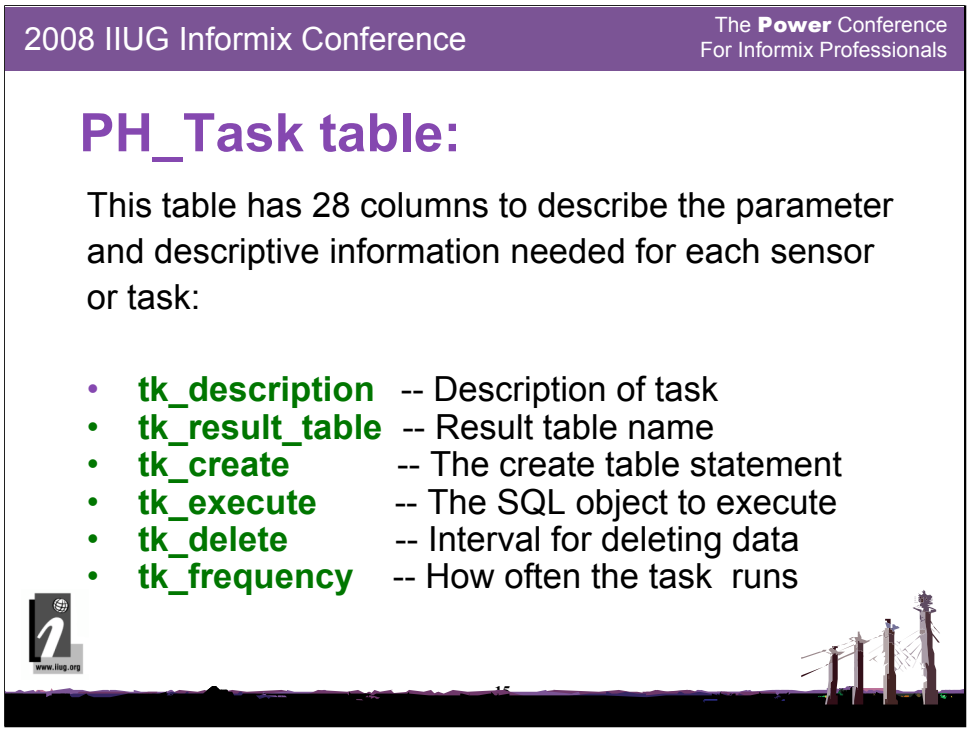
There are 7 'PH' Tables altogether that contain and organize the Scheduler task information.

The results tables have different names. The built-ins that come with the engine usually have names that start with 'mon_' as in 'monitor'.

The sysadmin tables are created in the rootdbs, and can be moved to a different dbspace if necessary.

The scripts that create these tables are found in $INFORMIXDIR/etc/sysadmin.

The definitions can be found in db_install.sql



All task and sensor functions you create will be stored in ph_task. The column names in green are the fields we can change. On the next couple slides there are a few other columns that the server updates for us.

tk_description lvarchar

tk_result_table varchar

tk_create lvarchar
tk_execute lvarchar
tk_delete interval day(2) to second
tk_frequency interval day(2) to second (max 99 days)

The tk_result_table column is used only by sensors and the content matches the table created in tk_create.

When the tk_delete interval is achieved, data is deleted from tk_result_table.

tk_create - The CREATE TABLE statement which will be executed to create the result table, if needed.

Note: The tk_create column is used by sensors. and as necessary, it is executed to create the table needed to hold the data for the sensor we create.

PH_Task table:

- **tk_id** - Sequential task ID
- **tk_name** - Unique Task name
- **tk_start_time** - Starting time of the task
- **tk_stop_time** - Stop time for a task
- **tk_type** - Type of Task
  - **TASK**
  - **SENSOR**
  - **STARTUP SENSOR**

Tk_id serial

TK_name char(36)

Tk_start_time datetime hour to second – when it is not a startup sensor

Tk_stop_time datetime hour to second – ignored if is a startup sensor

tk_type char(18)

Re: tk_type: Sensor = collects info

Startup Task – a task only executed when the server starts up

Startup Sensor – a sensor which is only executed at server startup.

undocumented -- STARTUP MONITOR -- present on 11.10, removed after.

PH_Task table:

- **tk_next_execution** - next time the task should be executed
- **tk_group** - Group name reference
- **tk_enabled** - scheduled or not scheduled ('t' = default)
- **tk_priority** - Job priority (1-5):
  0=Default
  5=execute first
  1=execute last

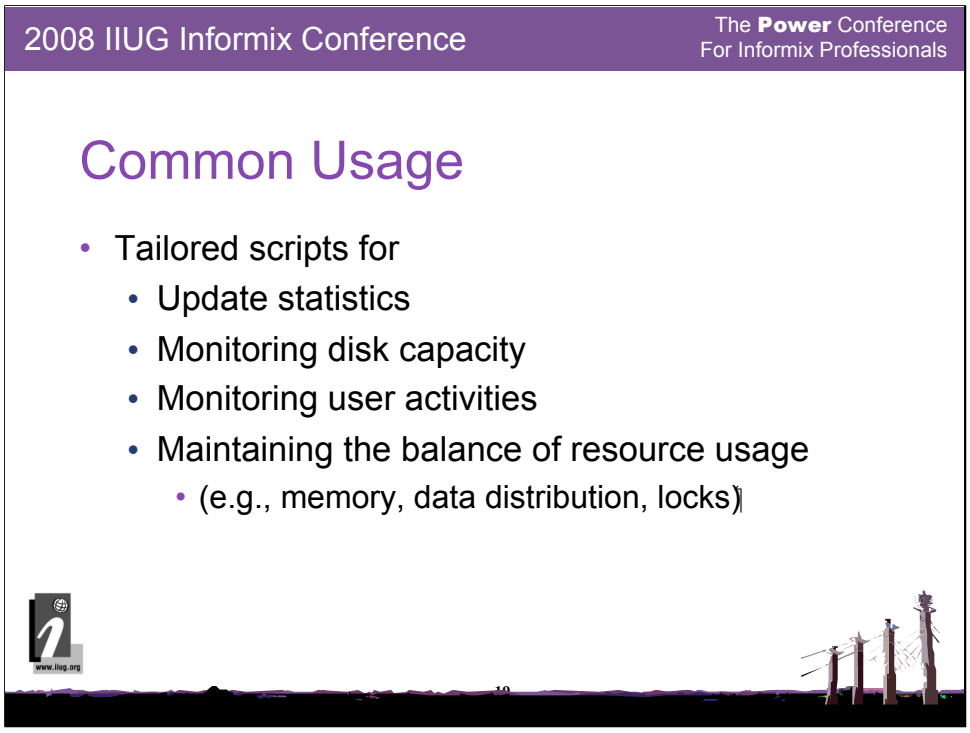
tk_next_execution datetime year to second
tk_group varchar(128) (ties to ph_group(group_name))
There are 11 predefined groups:
MISC,DISK,NETWORK,MEMORY,CPU,
TABLES,INDEXES,SERVER,USER,
BACKUP, and PERFORMANCE
(default is MISC).
tk_enabled boolean
tk_priority integer

ph_task table:

System controlled columns:

- **tk_sequence** - data collection number
- **tk_owner** - owner's thread ID
- **tk_attributes** - system flags for tracking status
- **tk_total_executions** - # of times to execute task
- **tk_total_time** - total time executing this task

Tk_sequence integer
Tk_owner integer
Tk_attributes integer
Tk_exec_num integer
Tk_exec_time integer



Before the Admin API came along, the choices were limited to options like the ones in this slide.

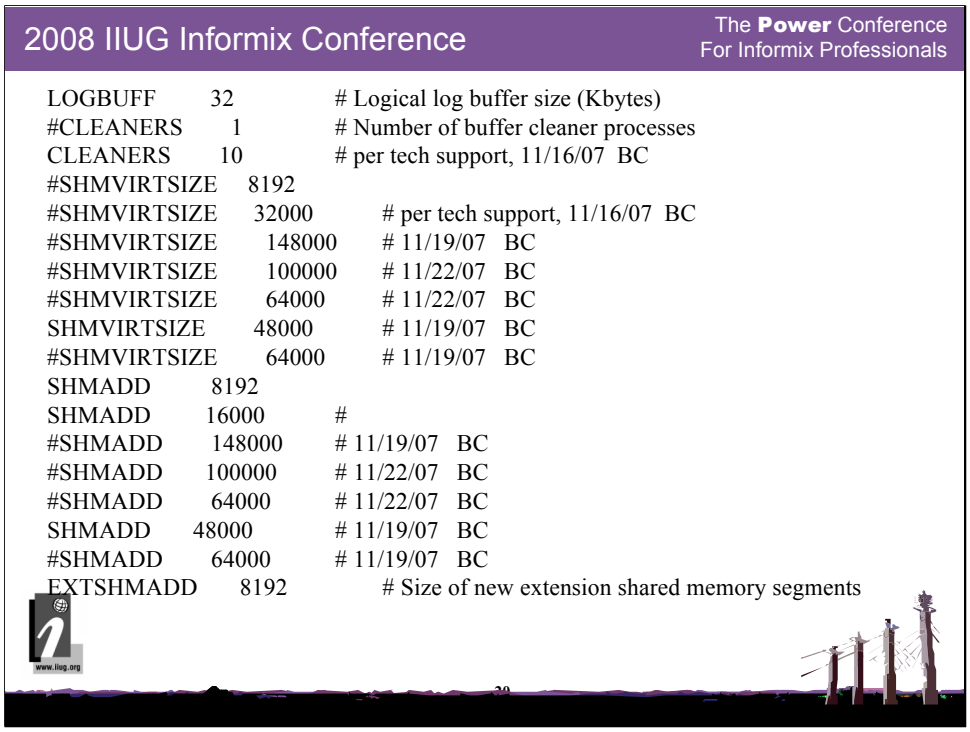
Some Guidelines if you want to use the API tools:

If you would like a graphical interface, consider using OAT – the Open Admin Tool.

(this requires some knowledge of php and web interfacing components)

If you need a complete Toolbox package for discovery, analysis and response remediation, consider using Sentinel or Server Studio.

If you need only one tool at a time, consider building the tool with the API



An example of what the built-in sensors will do for us.

Over time, our imaginary dba, “Big Chuck”, has been busy trying to fine tune his ONCONFIG file…

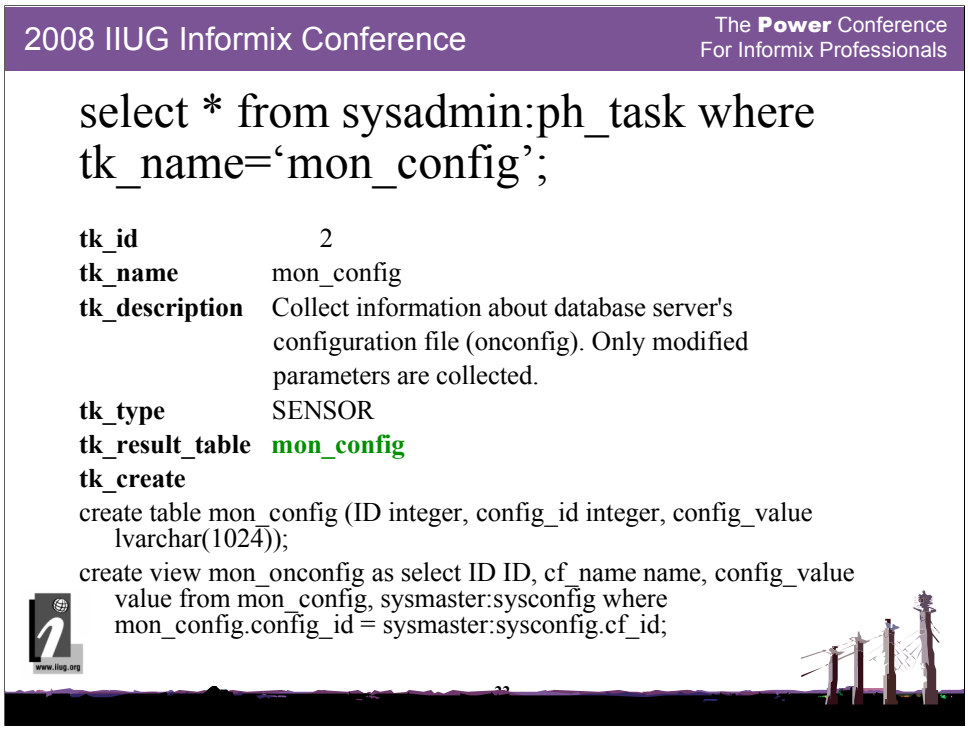
It is not pretty. At least he documents his changes.

A built-in sensor: mon_config

- ph_task comes with 13 sensors and startup sensors. One of these is **mon_config**.

[select * from sysadmin:ph_task]

(Note: The 11.5 release has 16 sensors)



Below is sample output from mon_onconfig.

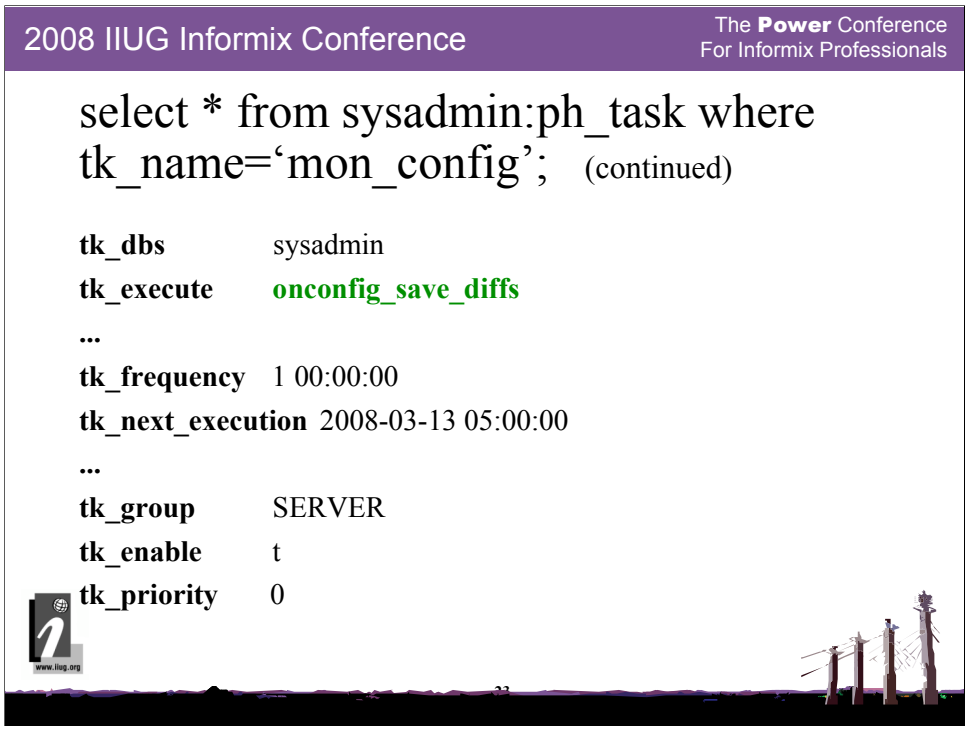
In this output (which is not dramatically readable by itself), we can surmise that SHMVIRTSIZE has changed twice since the engine was initialized. In a later example, we can tie in the ph_run output to find out exactly when the changes happened.

select * from mon_onconfig where name="SHMVIRTSIZE"

id 1
name SHMVIRTSIZE
value 0x0x2000

id 2
name SHMVIRTSIZE
value 0x0x7d00

id 78
name SHMVIRTSIZE
value 0x0x7d00



(The ellipsis on the slide above indicates there are a few fields I am not displaying).
Where is the function onconfig_save_diffs defined? ?

It is in $INFORMIXDIR/etc/sysadmin/sch_tasks.sql
(first created when the engine was initialized):

```
CREATE FUNCTION onconfig_save_diffs(task_id INTEGER, ID INTEGER)
  RETURNING INTEGER
 DEFINE value    LVARCHAR(1024);
 DEFINE conf_value  LVARCHAR(1024);
 DEFINE conf_id  INTEGER;
 LET value = NULL;
 FOREACH select cf_id, trim(cf_effective)
    INTO conf_id, conf_value
    FROM sysmaster:syscfgtab
 FOREACH  select FIRST 1 config_value
         INTO value
         FROM sysadmin:mon_config
         WHERE mon_config.config_id = conf_id
         ORDER BY id DESC
 END FOREACH
 IF conf_value == value THEN
     CONTINUE FOREACH;
 END IF
 INSERT INTO mon_config VALUES( ID, conf_id, conf_value );
 END FOREACH
 return 0;
END FUNCTION;
```

A Sensor example:

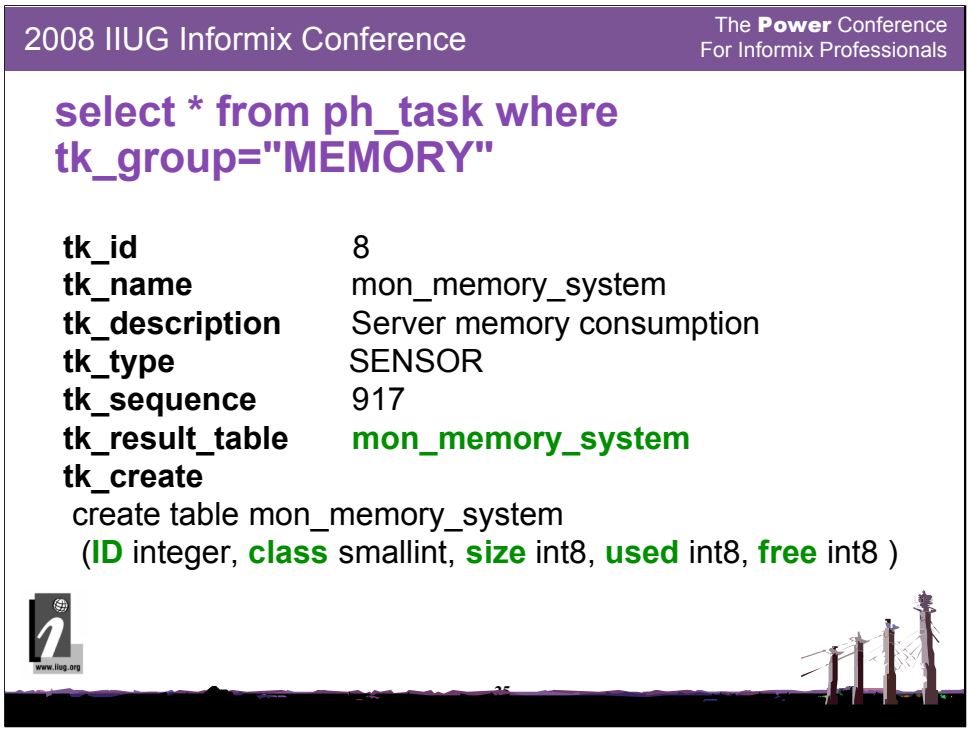
onstat -g seg

IBM Informix Dynamic Server Version 11.10.UC2 -- On-Line -- Up 28 days 07:20:12
-- 36480 Kbytes

Segment Summary:

id	key	addr	size	ovhd	class	blkused	blkfree
229378	1382107137	44000000	12189696	292924	R	2973	3
262147	1382107138	44ba0000	8388608	50184	V	2048	0
294916	1382107139	453a0000	8388608	50184	V	2033	15
327685	1382107140	45ba0000	8388608	50184	V	233	1815
Total:	-	-	37355520	-	-	7287	1833

Big Chuck notices a bunch of virtual segments, and decides he better get around to tuning his memory; he would like to use the new scheduler API.



What does he have in the tool box already?

The possible groups are:

MISC,DISK,NETWORK,MEMORY,CPU,TABLES,INDEXES,SERVER,USER,BACKUP,PERFORMANCE

select * from ph_task where tk_group="MEMORY“

Some things to notice

our results are going into a table named mon_memory_system

- the table has no dates

- if we want dates, we have to do a sql join with the PH_RUN table, using the tk_id

- the does have ‘class’, ’size’, ‘used’ and ‘free’ - similar to info from sysmaster:syssegments

- (class = 2 is the virtual class we are looking for)

select * from ph_task where tk_group="MEMORY“ (continued)

tk_dbs sysadmin
tk_execute insert into mon_memory_system select $DATA_SEQ_ID, seg_class, seg_size, seg_blkused, seg_blkfree FROM sysmaster:sysseglst
tk_delete 15 00:00:00
..
tk_frequency 0 02:00:00
tk_next_execution 2008-04-28 11:49:28

Notes:

-the execute statement for collecting data is grabbing from a new table named sysmaster sysseglst

(on old SMI queries we would have used sysmaster:syssegments)

-It collects 15 days of data before it recycles, and collects every 2 hours.

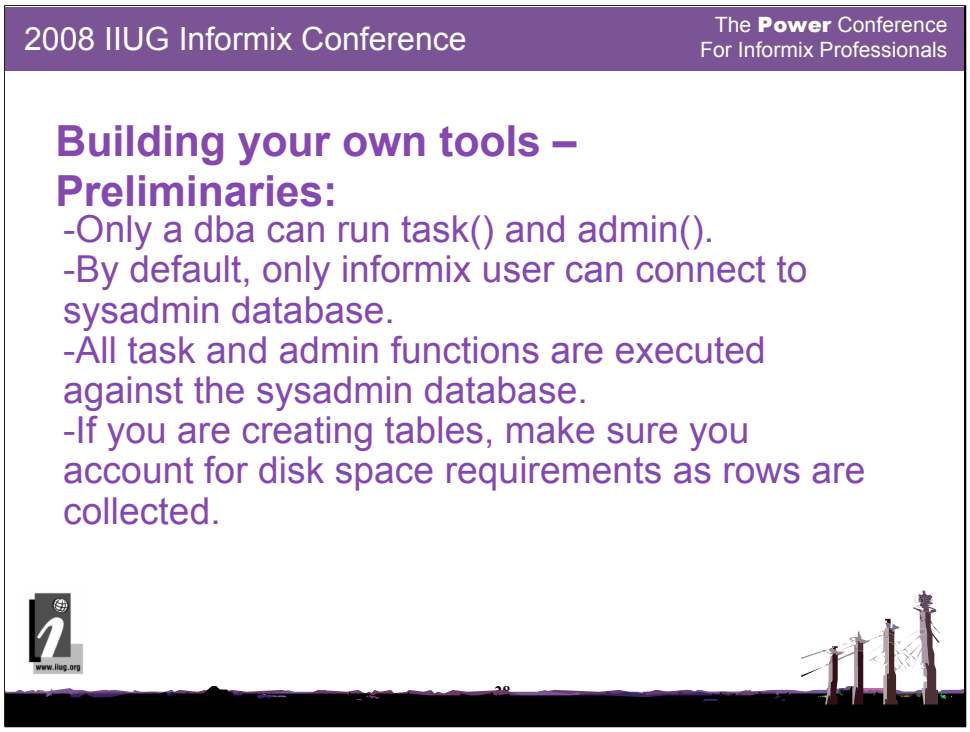
```
select date(a.run_time), b.size, b.used, b.free from
ph_run a, mon_memory_system b where class=2 and
b.id=a.run_id order by 1
```

(expression)	size	used	free
01/09/2008	8388608	2043	5
01/09/2008	8388608	533	1515
01/09/2008	8388608	2048	0
01/09/2008	8388608	2044	4
01/09/2008	8388608	533	1515
01/09/2008	8388608	2048	0
01/10/2008	8388608	2044	4
01/10/2008	8388608	532	1516
01/10/2008	8388608	2048	0
01/10/2008	8388608	2048	0
01/10/2008	8388608	2045	3
01/10/2008	8388608	531	1517
01/10/2008	8388608	2044	4
01/10/2008	8388608	2048	0
01/10/2008	8388608	532	1516

(For space considerations on this slide, I displayed date rather than datetime value.)

From the datetime references, Chuck was able to determine that he was averaging

3 virtual segments a day: he was able to correct this by setting SHMVIRTSIZE from 8192 to 25000.

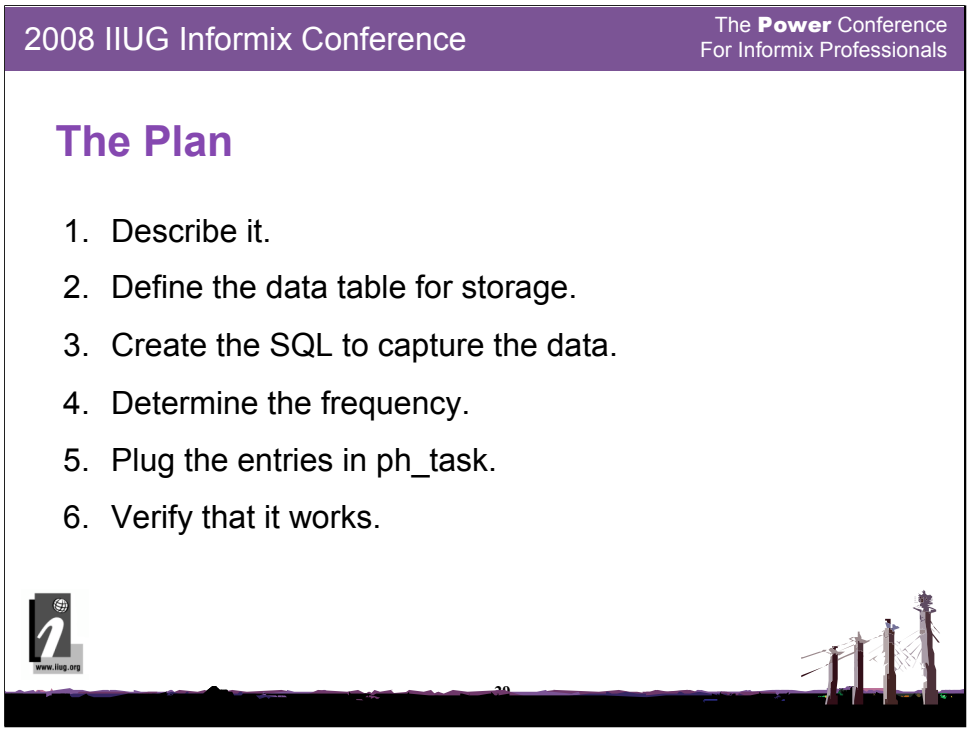


Preliminaries:

- Only a dba can run the task() and admin() functions.
- by default, informix is the only user that can connect to the sysadmin database.
- All task and admin functions are executed against the sysadmin database.

If you need to create a table, use this formula to estimate the amount of disk space needed:

(# of rows collected) * (size of the row collected) * (the # of records per day) * (the retention period).



To set up a task, you need to plan the task first. You need to have:

• A description of the task you want to monitor

• The table where you want to store data

• The SQL command, stored procedure, or function to capture data

• Information on when and how often you want the task to run, and how often to clear the data.

1. Describe it

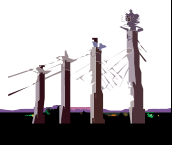
- **Monitor disk usage per database and dbspace.**

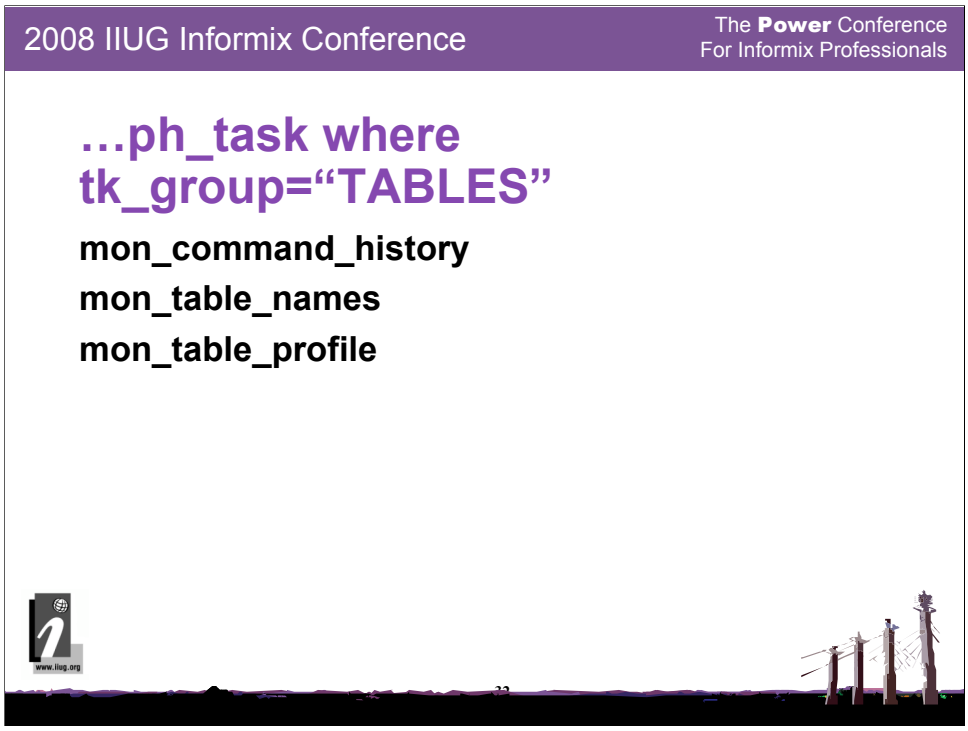
(DISK or TABLE group ?)

Do we have the tool already?

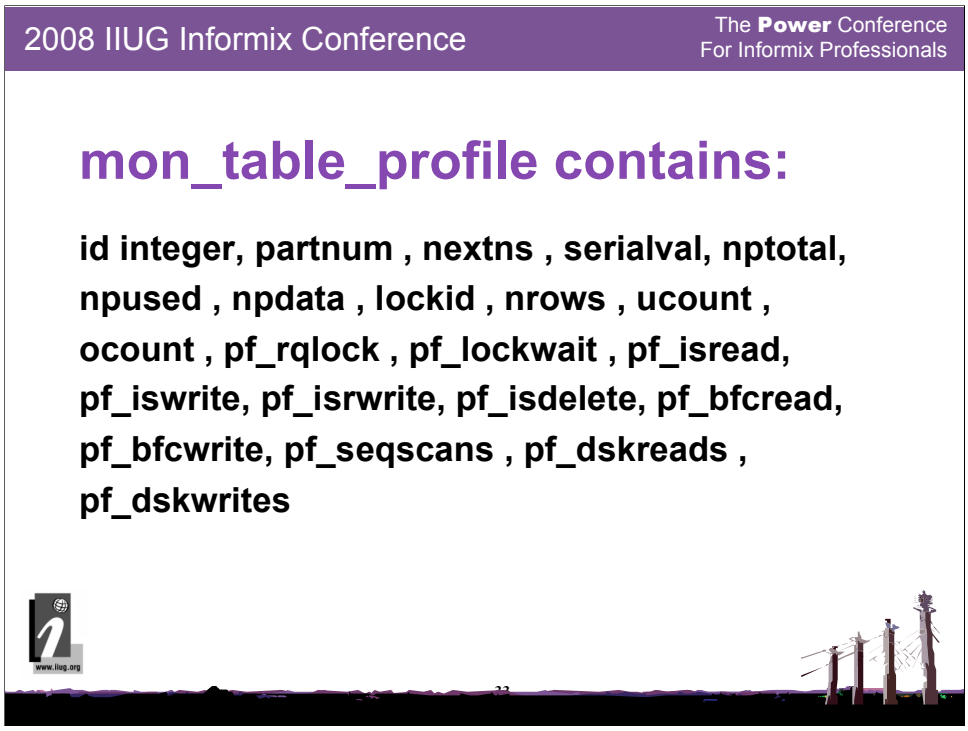
```
select count(*) from ph_task where
                                tk_group="DISK“
  count = 0

select count(*) from ph_task where
                                tk_group="TABLES“
  count =3
```





A closer look at ph_task to see what's in the tables group reveals that none of these tables appear to relate to what we need.



A closer look at mon_table_profile shows it tracks pages and i/o usage, but nothing we can use.

So, we will want to create our own table for tracking.

2. Define the data table for storage

Track:

- **The dbspace by name**
- **The database by name**
- **Total space used per database**
- **Number of extents per database**

2. Define the data table for storage

dbspace	database	tot_space	no_of_exts
dbs2	stores7	1263	133
rootdbs	most	1100	103
rootdbs	sysadmin	2232	212
rootdbs	sysmaster	1388	137
rootdbs	sysuser	1072	102
rootdbs	sysutils	1136	114

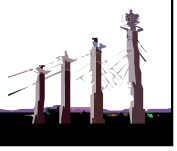
(A sample of what I want to monitor)

2. Define the data table for storage

```
create table mon_dbs_use (
ID integer,
dbspace char(10),
database char(10),
tot_space integer,
no_exts integer)
```

3. Create the SQL to capture the data

```
select distinct
    substr(DBINFO("DBSPACE",partnum),1
    ,10) DBSpace, dbsname[1,10] Database,
    sum(pe_size) tot_space, count(*)
    no_of_exts
from sysmaster:sysptnext,
    sysmaster:systabnames
where pe_partnum = partnum
and tabname != "TBLSpace"
group by 1,2
```



This is the query which I know will capture the data visible in the previous slide.

4. Determine the frequency

- **STARTUP SENSOR**
- **Delete every 90 days**

If this were a regular sensor, I would specify ‘SENSOR’, and a frequency interval at which to run.

5. Insert the entries in ph_task

```
Insert into ph_task
(tk_name,    tk_description,    tk_type,
tk_result_table,   tk_create,   tk_dbs,
tk_execute, tk_delete, tk_frequency,
tk_group, tk_enable)
VALUES   …
```

5. Insert the entries in ph_task (continued)

```
VALUES
('mon_dbs_use',
'Monitor disk usage per database and dbspace',
'STARTUP SENSOR','mon_dbs_use',
'create table mon_dbs_use (ID integer, dbspace
char(10),database char(10), tot_space integer,no_exts
integer)','sysadmin','insert into mon_dbs_use select
$DATA_SEQ_ID, distinct substr(DBINFO(DBSPACE,par
tnum),1,10) DBSpace, dbsname[1,10]
Database,sum(pe_size) tot_space, count(*) no_of_exts  from
sysmaster:sysptnext, sysmaster:systabnames where
pe_partnum = partnum and tabname != TBLSpace group by
1,2','90 00:00:00', '0 00:01:00','DISK','t')
```

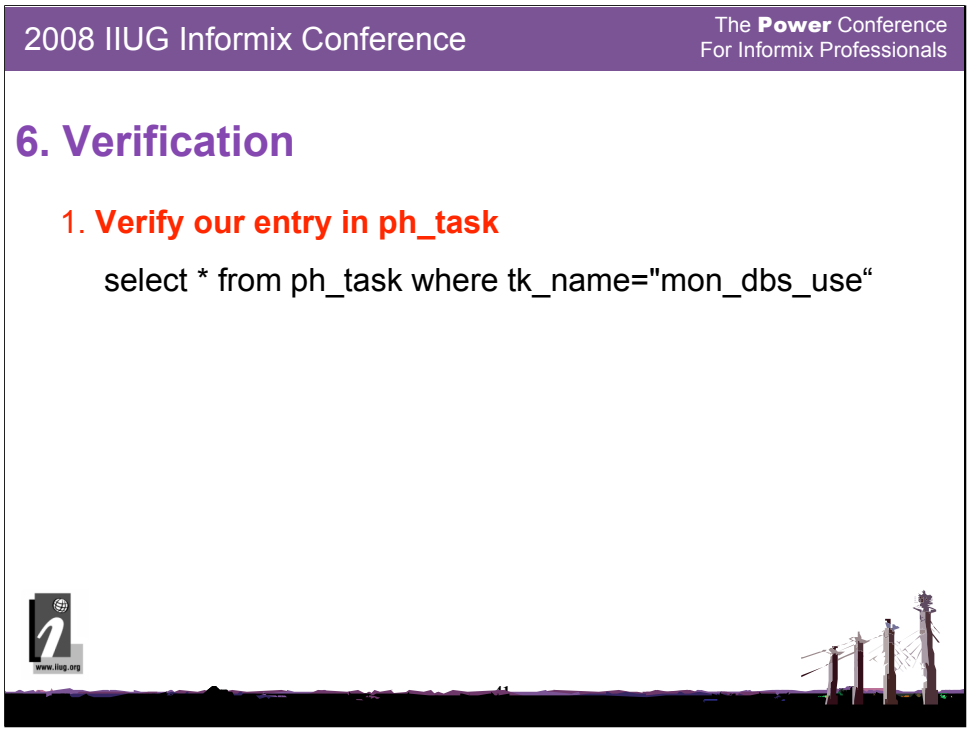
1. When inserting your sql into the tk_execute column, if you have too much hardship getting the

SQL expression(in blue) to work because of misplaced quotes or other errors, create the sql as

a standalone function, then insert the name of the function to be called in the tk_execute column.

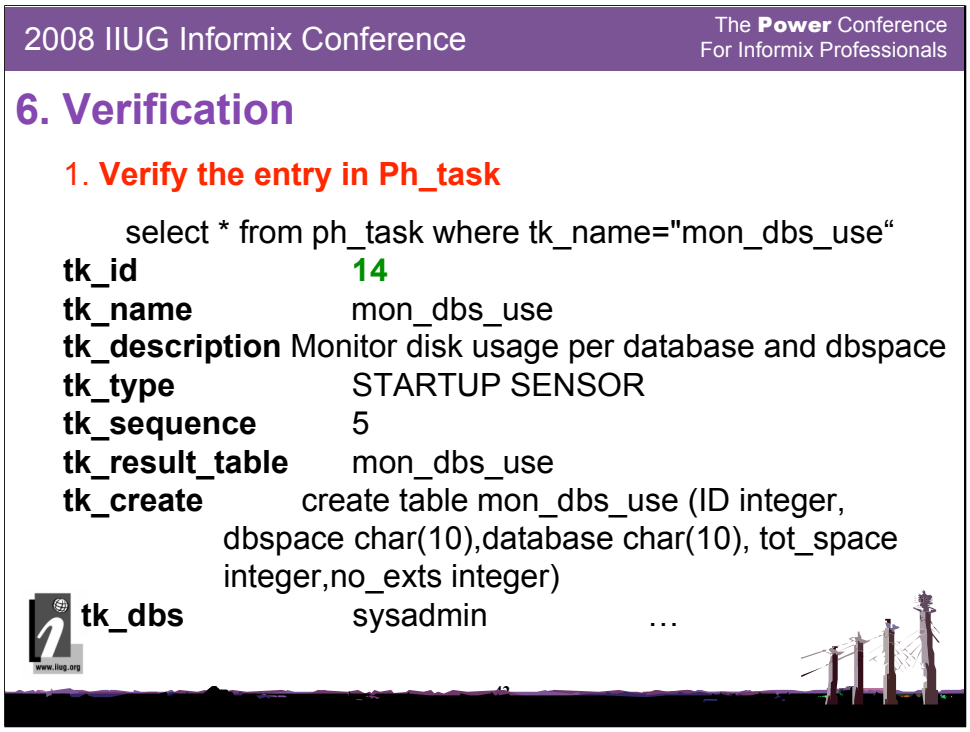
2. Always include a $DATA_SEQ_ID in your tk_execute process, and an integer ID column in your result

table, so you can keep track of the run_task sequence.

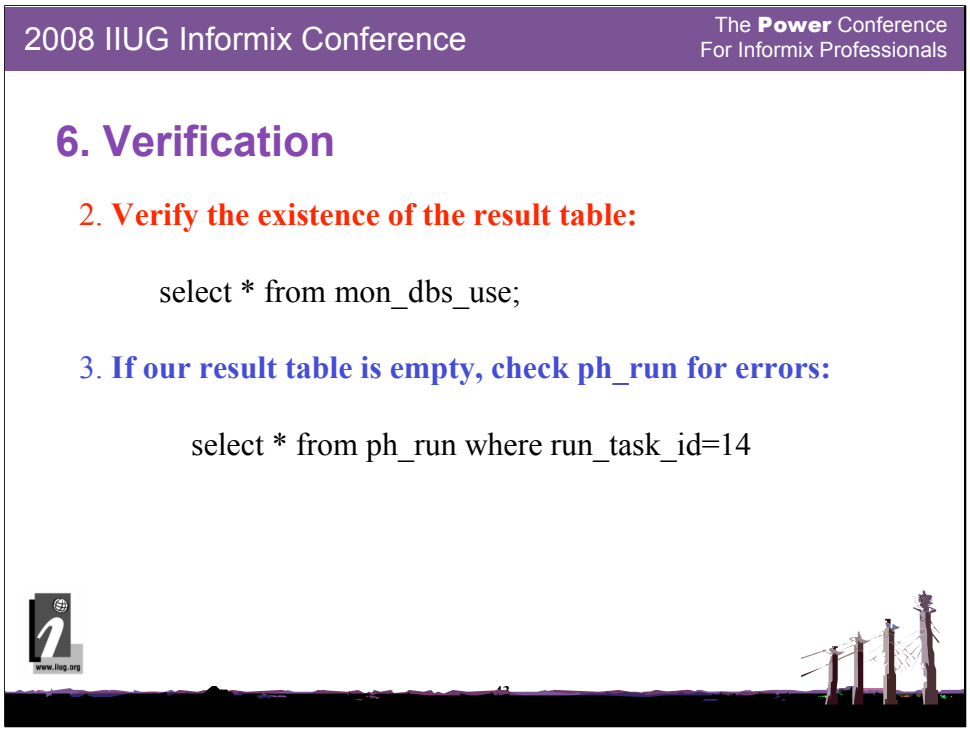


When out task is inserted, it is also executed in order to create the result table.

In our first verification step, we are verifying insert syntax…
We should get back the row that we just put in:
select * from ph_task where tk_name="mon_dbs_use“;



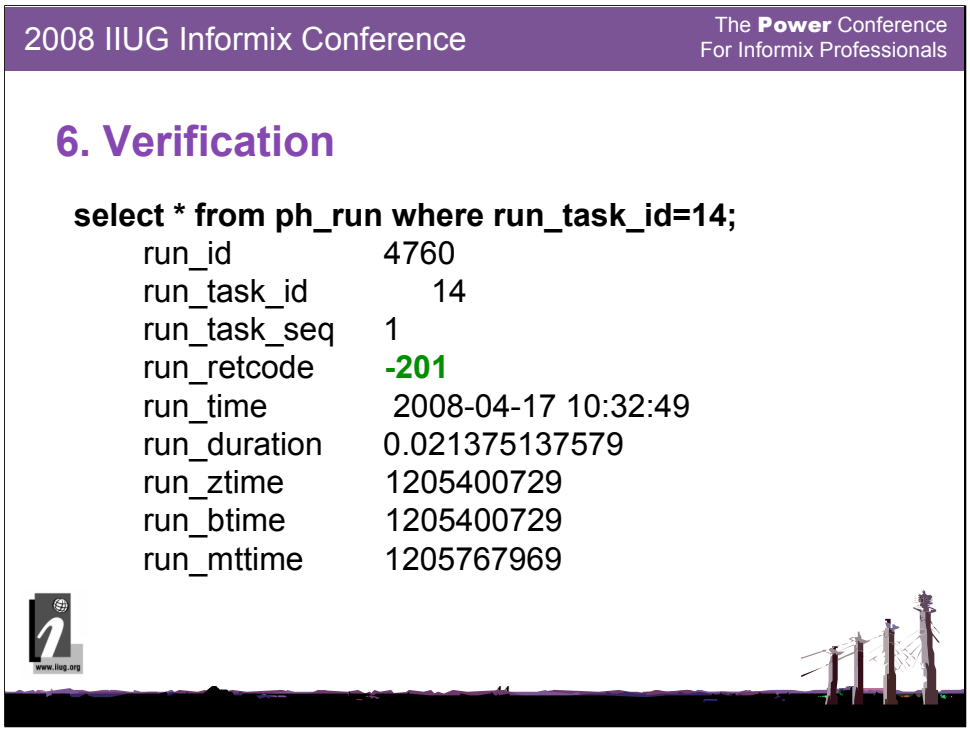
Note the tk_id assigned for our serial column: 14. (We will use this for the next verification step.)



2. **Verify the existence of the result table:**

select * from mon_dbs_use;

If the table does not exist or it has no rows, we have a syntax error in the ph_task table.



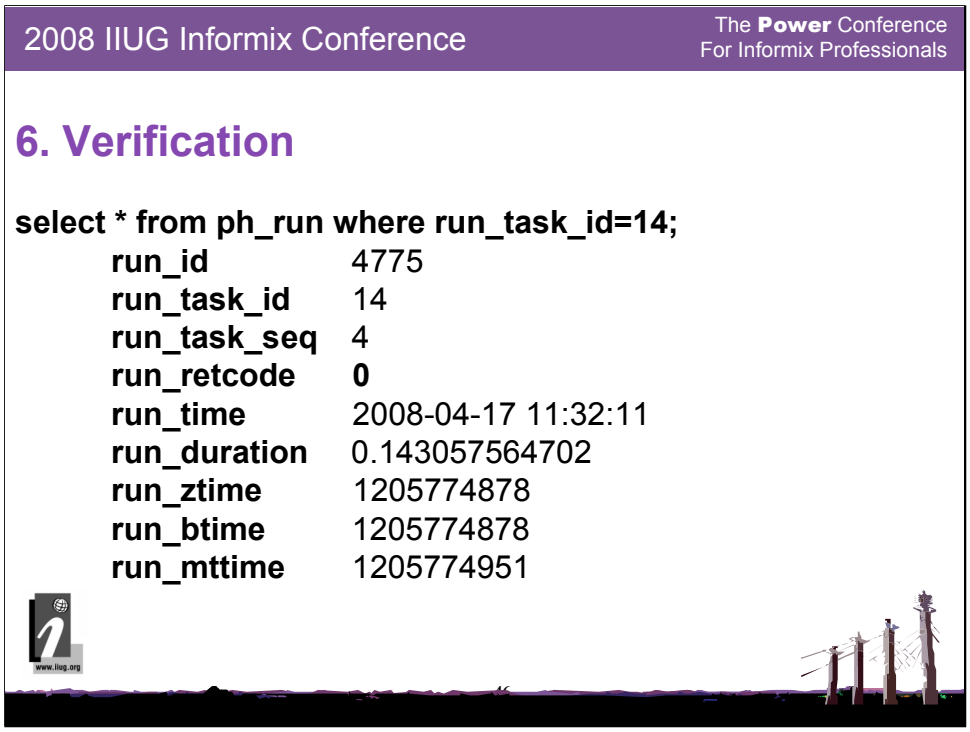
Selecting from the ph_run table for our task id, we find that there is a -201 (syntax error) in our executed statement (tk_execute has a problem).

(fix for the values clause used on step 5)

update ph_task set tk_execute= "insert into mondbs_use select distinct $DATA_SEQ_ID, substr(DBINFO('DBSPACE',partnum),1,10) DBSpace, dbsname[1,10] Database, sum(pe_size) tot_space, count(*) no_of_exts from sysmaster:sysptnext, sysmaster:systabnames where pe_partnum = partnum and tabname != 'TBLSpace' group by 2,3" where tk_name="mon_dbs_use"

Be careful with the insertion clauses. Use double quotes to bracket the whole phrase,

single quotes for the parts inside.



After an update for our startup sensor, our tk_execute column is now fixed – but the task is not going to run automatically, since it is not newly instantiated.

To verify it at this point, we have to bounce the engine.

After the bounce, a check of the ph_run table will now show a 0 return code.

6. Verification

```
select * from mon_dbs_use;
```

id	dbspace	database	tot_space	no_exts
1	dbs2	stores7	1263	133
1	rootdbs	most	1100	103
1	rootdbs	sysadmin	1496	147
1	rootdbs	sysmaster	1388	137
1	rootdbs	sysuser	1072	102
1	rootdbs	sysutils	1136	114

And our mon_dbs_use table has data..

Q&A
?

Resources

- **IBM Informix Dynamic Server Administrator's Reference**
- **IBM Informix Guide to SQL: Syntax**
- **"Informix Dynamic Server 11:Advanced Functionality for Modern Business" a redbook located at** http://www.redbooks.ibm.com/abstracts/sg247465.html?Open&pdfbookmark
- **Certification Tutorial, part 2.**
- **Focused summary on monitoring at**

  http://www.serverstudio.com/

Session D12
Sensor & Task functions in IDS 11

David Jay
IBM Corporation
djay@us.ibm.com