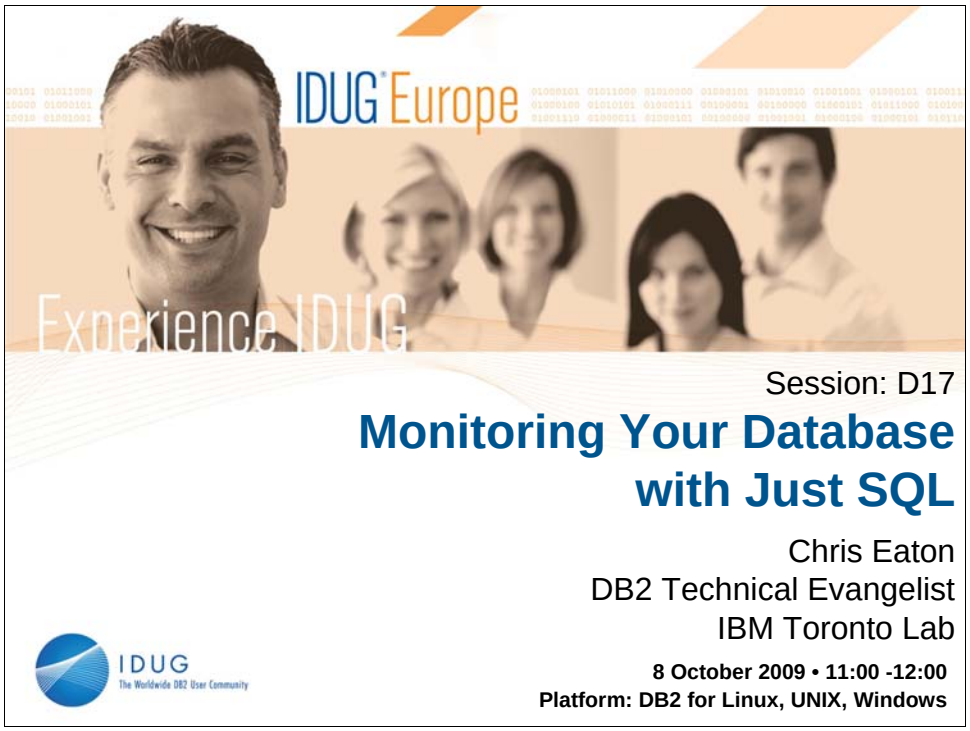
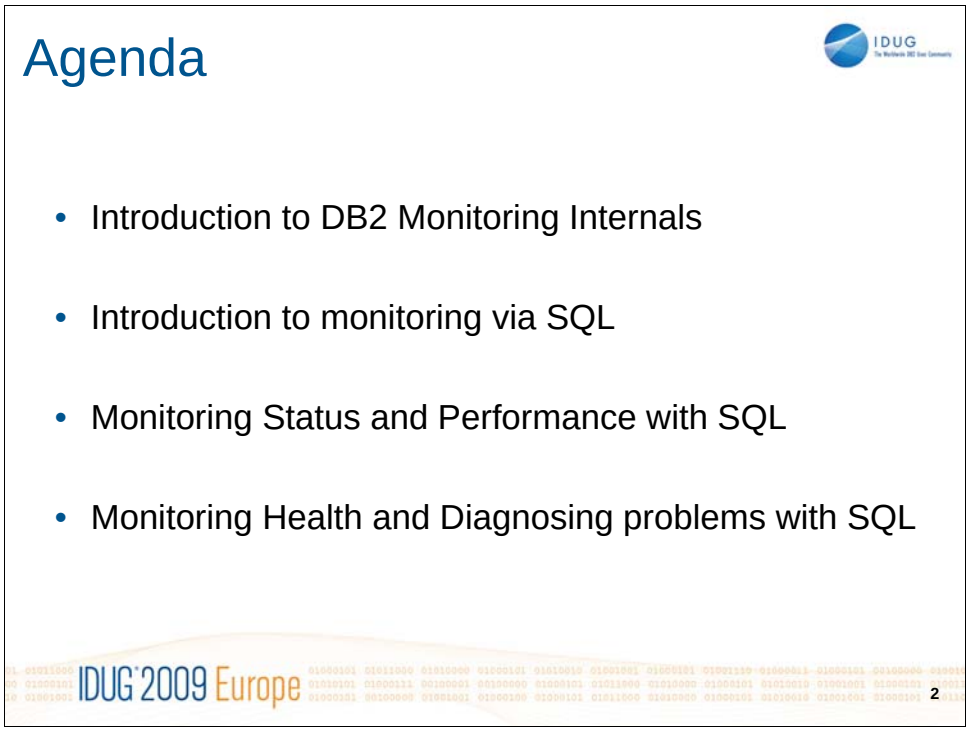


Abstract:

DB2 9 now offers very useful administrative views that are built into the server. These views can help monitor database status, system performance, database health as well as help diagnose database problems. The big benefit is that you can do this with simple SQL statements in your very own simple scripts. Chris will take you through these new administrative views and you will walk away with a set of scripts to get you started.



Outline:

DB2 9 now offers very useful administrative views that are built into the server. These views can help monitor database status, system performance, database health as well as help diagnose database problems. The big benefit is that you can do this with simple SQL statements in your very own simple scripts. Chris will take you through these new administrative views and you will walk away with a set of scripts to get you started.

Objectives:

Introduction to the DB2 9 administrative views

Queries to monitor database status

Queries to monitor database performance

Queries to monitor database health

Queries to diagnose database problems

Introduction to DB2 Monitoring Internals

DB2 Monitoring Internals

- What is Snapshot monitoring?
  - A “picture” of the state of the DB2 system at a point in time
  - A report on a set of counters (mostly) stored inside DB2
  - Just like a camera, a snapshot is initiated by a human
- What is an Event monitor?
  - A similar set of information (counters mostly) triggered by a defined event
  - For example, information about what an application did when it disconnects from the database
  - We won’t discuss Event Monitoring in this session

Snapshot monitoring gives you a view into what is happening with the database at an instant in time. Just like a photograph captures a speeding car driving down the road, so does the snapshot capture constantly changing information that is being tracked by DB2. By the time you look at the snapshot, the values will likely have changed. The way a snapshot is triggered is by external interfaces. So as an administrator, you need to run a sql statement or call an application programming interface (API) to see the snapshot information.

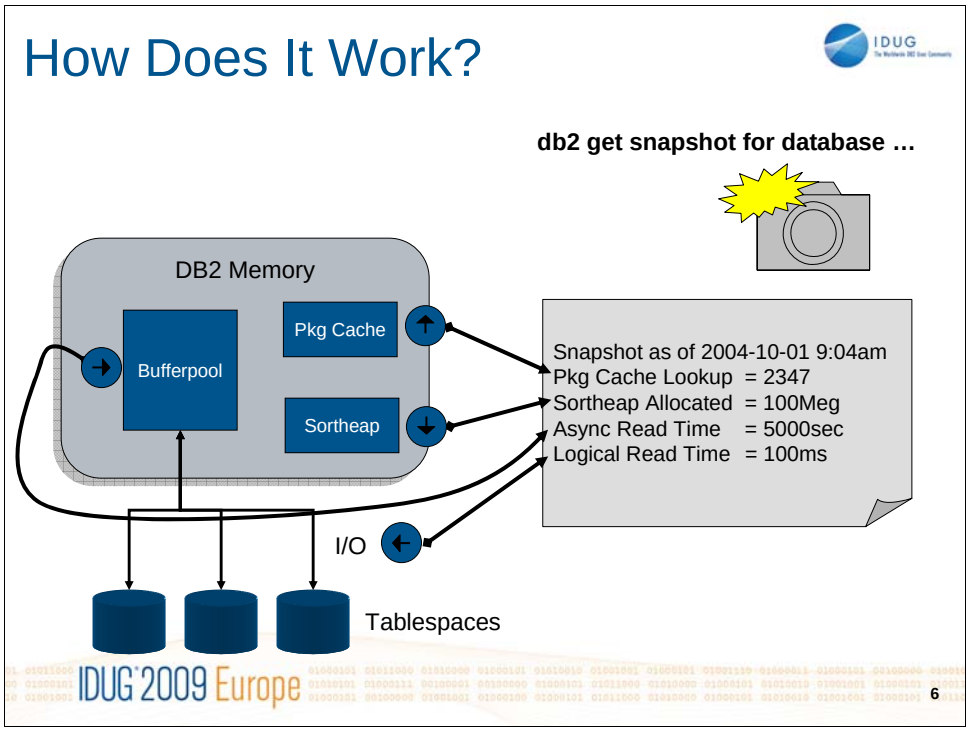
An event monitor on the other hand shows you what is happening when a given event occurs. The collection of this information is triggered by an internal event. For example, you can set an event monitor to trigger whenever an application disconnects or when a deadlock occurs. Most of the elements gathered are the same for both snapshot and event monitors.

In the rest of this presentation we will focus only on snapshot monitoring.

Types of Monitor Elements

- **Counters**
  - Measures the number of times an activity occurs (always increases) – Can be reset
  - E.g.: Rows read from a table, number of physical page reads, etc.
- **Gauges**
  - Indicates the current value of an item (may increase or decrease over time) – not reset (value are current state)
  - E.g.: Number of currently active sorts, amount of log space currently allocated, etc.
- **Information**
  - Reference type information about a monitor element – not reset
  - E.g.: Server Platform, Authentication ID of connected user, etc.
- **Timestamp**
  - Indicates the date and time an activity took place. – not reset. Number of seconds and microseconds since Jan 1, 1970
  - E.g.: Last time a database was backed up, snapshot time, etc.
- **Time**
  - Returns the number of seconds and microseconds spent on an activity – Can be reset
  - E.g.: Time spent reading data pages, elapsed time of a unit of work, etc.

There are roughly 540 monitor elements in total for DB2. Elements can be categorized into the above types. Some give you counts, others give you high water marks while others give you timing information and/or general information. In the DB2 System Monitor Guide and Reference you can see a description of all 500+ monitor elements, what they store and how you can access them.



There are a large number of monitor elements keeping track of what is going on in the database. Most of the monitor elements are like counters that continue to increment. For example the number of lock escalations encountered, number of package cache lookups, rows read from a table, rows written to a table, etc. Other counters act as a high water mark like maximum total log space used, maximum number of concurrent connections, etc. And others are more of informational like log object names (for lock monitors), userid of the user running an application, etc.

When a snapshot is taken, the current value of these “counters” is displayed to the user.

Counters can be reset at any time (back to 0 or null) and are automatically reset when the instance is restarted.

Command Line Syntax

- GET SNAPSHOT FOR
  - DATABASE MANAGER
  - DATABASE ON <dbname>
  - TABLESPACES ON <dbname>
  - TABLES ON <dbname>
  - BUFFERPOOLS ON <dbname>
  - LOCKS ON <dbname>
  - APPLICATIONS ON <dbname>
  - DYNAMIC SQL ON <dbname>
- You must have SYSADM, SYSCTRL, SYSMAINT or SYSMON authority

In addition to the above, you can also narrow in on specific applications by running a get snapshot for application and specifying the specific application id that you want to get information on.

Details on the exact syntax can be found in both the DB2 System Monitor Guide and Reference as well as in the DB2 Command Reference.

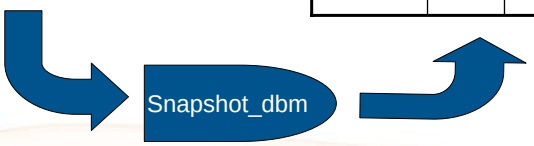
Introduction to Monitoring via SQL Functions

What's a Table UDF

- UDF = User Defined Function
  - Shipped with DB2 – not user defined
- A function that takes a structured set of information and makes appear to be a table

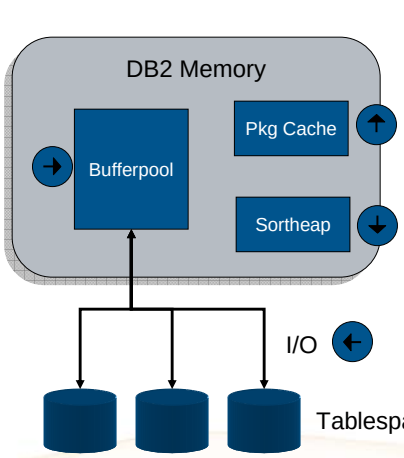
```
Instance name                       = DB2
Database manager status             = Active
Service level                       = s040219
Private Sort heap allocated         = 0
Private Sort heap high water mark   = 277
```

Instance_ name	Status	Serv_l evel	Priv_sor t_alloc	Priv_sor t_high
DB2	Active	S0402 19	0	277



The term User Defined Function is somewhat of a misnomer in this case because these UDFs are shipped by IBM (there is no user definition required). The set of monitor UDFS take the monitor information and displays the information as if it were a table to the rest of DB2. By turning the monitor information into table data, you can then use the full power of SQL to manipulate and present the data however you want. You can select only specific columns, perform arithmetic on the values, etc. You will see a lot of examples that show what is possible in the next set of slides.

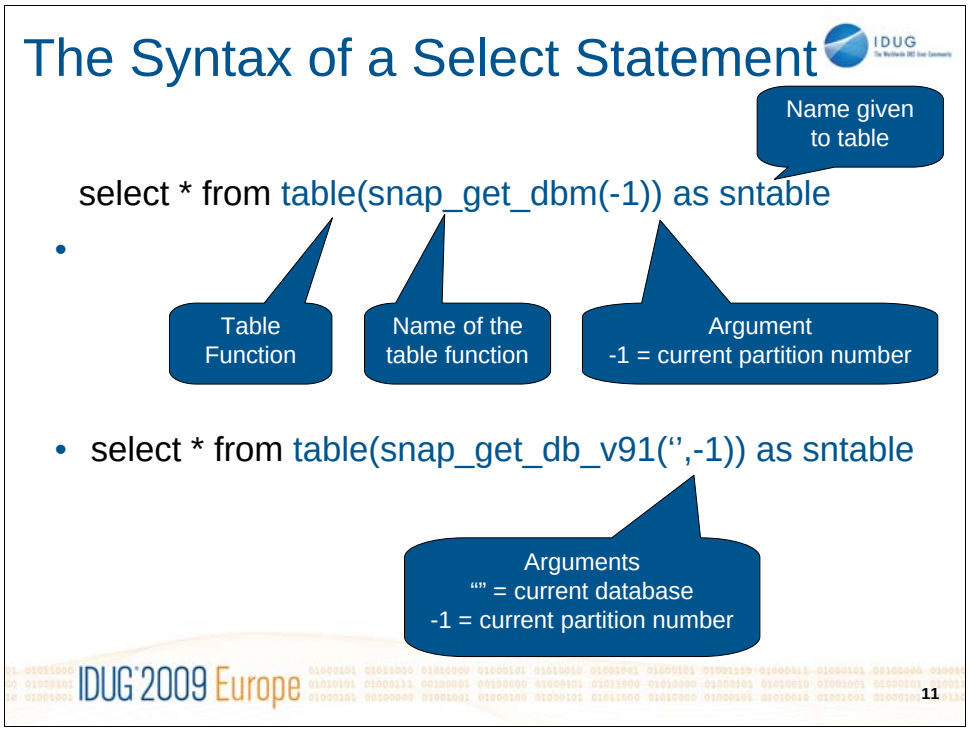
How Does It Work?



SELECT * FROM TABLE(SNAP_GET_DBM)

Snapshot_time	Pkg_cache_lookup	Sortheap_alloc	Async_read_time	Logical_read_time
2004-10-01 9:04am	2347	100	5000	100

Executing the select statement works just like the clp command in that the monitors are read and presented to the user when the snapshot is taken



The syntax may look a bit strange because it is using the user defined table function standard syntax. Each snapshot udf function takes either 1 or 2 arguments. Those that take a single argument are at the instance level (so you don't need to specify a database name). The argument specifies the database partition number that you want the information to come from. A -1 indicates to take the information from the database partition that you are currently connected to.

For those udfs that take 2 arguments, the first is the database name that you want the snapshot info from. If you specify a null, then the information is selected from the currently connected database. The second argument is the database partition number.

DB2 9 Makes Your Life Simpler – Administrative Views

- Table Functions still exist but now you have VIEWS
- All views are in the SYSIBMADM schema
- Convert coded values to text strings
- Can be a control point to allow people with lower authority to view monitor information
  - Grant select on view and execute on table function

By using the new administrative views for all the snapshot table functions, your SQL can become a bit easier to read and write.

To select all the columns out of the snapshot_database udf, you need to run

```
select * from table(snapshot_database(‘’,-1)) as sntable
```

However, if you use the SYSIBMADM.SNAPDB view the above select statement becomes

```
select * from sysibmadm.snapdb
```

So much less typing. As you will see in the next section, this simplification makes it much easier to read your SQL.

SNAPSHOT Views

- **Database Manager**
  - SNAPDBM
  - SNAPDBM_MEMORY_POOL
- **Database Level**
  - **SNAPDB**
  - SNAPDB_MEMORY_POOL
  - SNAPBP
  - SNAPBP_PART
  - SNAPHADR
- **Application Level**
  - **SNAPAPPL**
  - **SNAPAPPL_INFO**
  - **SNAPLOCKWAIT**
  - SNAPSTMT
  - SNAPAGENT
  - SNAPSUBSECTION
  - SNAPAGENT_MEMORY_POOL
  - SNAPDYN_SQL
  - SNAPLOCK
- **Object Level**
  - SNAPTAB
  - SNAPTAB_REORG
  - SNAPTBSP
  - SNAPTBSP_PART
  - SNAPTBSP_QUIESCER
  - SNAPCONTAINER
  - SNAPTBSP_RANGE
  - SNAPUTIL
  - SNAPUTIL_PROGRESS
  - SNAPDETAILLOG
  - SNAPSTORAGE_PATHS
- **Database Partitioning Feature (DPF)**
  - SNAPFCM
  - SNAPFCM_PART

New items are being added every release. The items above include views in all DB2 8 and 9 releases. The ones in bold are the ones we will discuss further in this presentation

"Convenience" Monitor Views

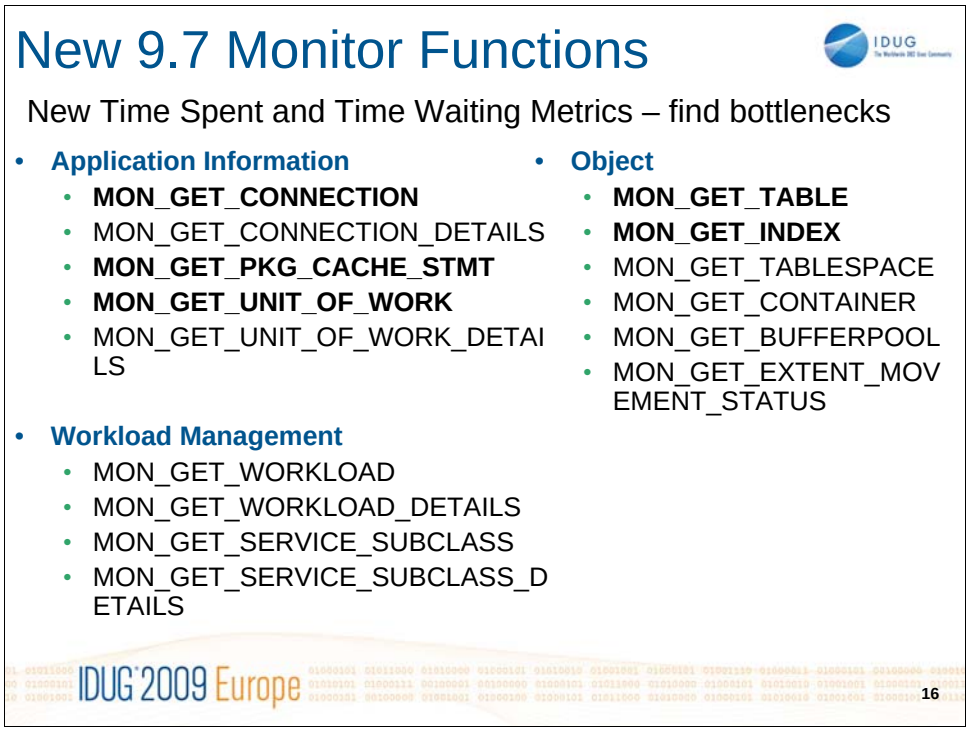
- APPLICATIONS
- APPL_PERFORMANCE
- **BP_HITRATIO**
- BP_READ_IO
- BP_WRITE_IO
- CONTAINER_UTILIZATION
- LOCKS_HELD
- LOCKWAIT
- LOG_UTILIZATION
- **LONG_RUNNING_SQL**
- QUERY_PREP_COST
- TBSP_UTILIZATION
- TOP_DYNAMIC_SQL

In DB2 9 some additional views were created for your convenience. That is, they are based off of the views in the previous chart but they make calculations easier by pre-computing some information for you. For example, rather than having to calculate hit ratios by dividing logical reads by (logical reads plus physical reads), the BP_HITRATIO view does that for you.

Administrative Views

- ADMINTABINFO
- ADMINTABCOMPRESSINFO
- ADMIN_GET_INDEX_INFO
- ADMIN_GET_INDEX_COMPRESS_INFO
- ADMIN_EST_INLINE_LENGTH
- ADMIN_IS_INLINED
- ADMIN_GET_DBP_MEM_USAGE
- DBCFG
- DBMCFG
- REG_VARIABLES
- DB_PARTITIONS
- **DB_HISTORY**

In addition to the convenience views, there are a set of administration views that can assist administrators in finding specific information. For example, you can see table compression information (percent of compression, space saved, etc), registry variables directly from SQL and much more.



DB2 9.7 is adding even more monitoring capabilities by instrumenting DB2 internals even further. These monitor table functions allow you to look at how much time was spent processing specific pieces of the query (cpu time, compile time, sort time, read time, etc) as well as how much time is spent waiting for things (waiting on I/O, logging, locks, etc.).

Monitoring Performance With SQL Select Statements

Long Running SQL


```
SELECT ELAPSED_TIME_MIN,
       SUBSTR(AUTHID,1,10) AS AUTH_ID,
       AGENT_ID,
       APPL_STATUS,
       SUBSTR(STMT_TEXT,1,20) AS SQL_TEXT
FROM   SYSIBMADM.LONG_RUNNING_SQL
WHERE ELAPSED_TIME_MIN > 0
ORDER BY ELAPSED_TIME_MIN DESC
```

```
ELAPSED_TIME_MIN AUTH_ID  AGENT_ID APPL_STATUS SQL_TEXT
---------------- -------- -------- ----------- --------------
               6 EATON         878  LOCKWAIT     update org set deptn
```


A very easy to use administrative view is the SYSIBMADM.LONG_RUNNING_SQL view which can quickly show you the longest running SQL statements currently executing in your database. The columns of interest are below

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP		TIMESTAMP Time the report was generated.
ELAPSED_TIME_MIN	INTEGER	Elapsed time of the statement in minutes.
AGENT_ID	BIGINT	Application Handle (agent ID)
APPL_NAME	VARRHAR(256)	Application Name
APPL_STATUS	VARCHAR(22)	Application Status.
AUTHID	VARCHAR(128)	Authorization ID
INBOUND_COMM_ADDRESS		VARCHAR(32) Inbound Communication Address
STMT_TEXT	CLOB(16 M)	SQL Dynamic Statement Text
DBPARTITIONNUM	SMALLINT	The database partition from which the data was retrieved for this row.

Buffer Pool Query

- Display buffer pool hit ratios (data, index and XML)

```
SELECT      SUBSTR(BP_NAME,1,20) as BP_NAME,
            TOTAL_HIT_RATIO_PERCENT as ALL_HR,
            DATA_HIT_RATIO_PERCENT as DATA_HR,
            INDEX_HIT_RATIO_PERCENT as INX_HR,
            XDA_HIT_RATIO_PERCENT as XML_HR
FROM SYSIBMADM.BP_HITRATIO;
```

```
BP_NAME              ALL_HR  DATA_HR INX_HR  XML_HR
-------------------- ------- ------- ------- -------
IBMDEFAULTBP             98      80      99       0
LARGE_BP                 99      99       0       0
SMALL_BP                 25      25       0       0
```


As previously mentioned the BP_HITRATIO view makes it much easier to write SQL to monitor key bufferpool metrics. Here are the other columns in this view:

Column name	Data type	Description or corresponding monitor element
SNAPSHOT_TIMESTAMP	TIMESTAMP	Timestamp when the report was requested.
DB_NAME	VARCHAR(128)	db_name - Database name
BP_NAME	VARCHAR(128)	bp_name - Buffer pool name
TOTAL_LOGICAL_READS	BIGINT	Total logical reads (index, XDA and data) in the bufferpool.
TOTAL_PHYSICAL_READS	BIGINT	Total physical reads (index, XDA and data) in the bufferpool.
TOTAL_HIT_RATIO_PERCENT	DECIMAL(5,2)	Total hit ratio (index, XDA and data reads).
DATA_LOGICAL_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
DATA_PHYSICAL_READS	BIGINT	pool_data_p_reads - Buffer pool data physical reads
DATA_HIT_RATIO_PERCENT	DECIMAL(5,2)	Data hit ratio.
INDEX_LOGICAL_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
INDEX_PHYSICAL_READS	BIGINT	pool_index_p_reads - Buffer pool index physical reads
INDEX_HIT_RATIO_PERCENT	DECIMAL(5,2)	Index hit ratio.
XDA_LOGICAL_READS	BIGINT	pool_xda_l_reads - Buffer Pool XDA Data Logical Reads
XDA_PHYSICAL_READS	BIGINT	pool_xda_p_reads - Buffer Pool XDA Data Physical Reads
XDA_HIT_RATIO_PERCENT	DECIMAL(5,2)	Auxiliary storage objects hit ratio.
DBPARTITIONNUM	SMALLINT	The database partition from which the data for the row was retrieved.

Package Cache Query

- Look at all the queries in the package cache
  - Both Dynamic and Static
  - See execution time, wait time (by component), and much more

```
SELECT
   SUBSTR(STMT_TEXT,1,20) AS STMT,
   SECTION_TYPE AS TYPE,
   NUM_EXECUTIONS,
   TOTAL_ACT_TIME AS TOTAL_TIME,
   TOTAL_ACT_WAIT_TIME AS WAIT_TIME
FROM TABLE(MON_GET_PKG_CACHE_STMT('','',-1))
```

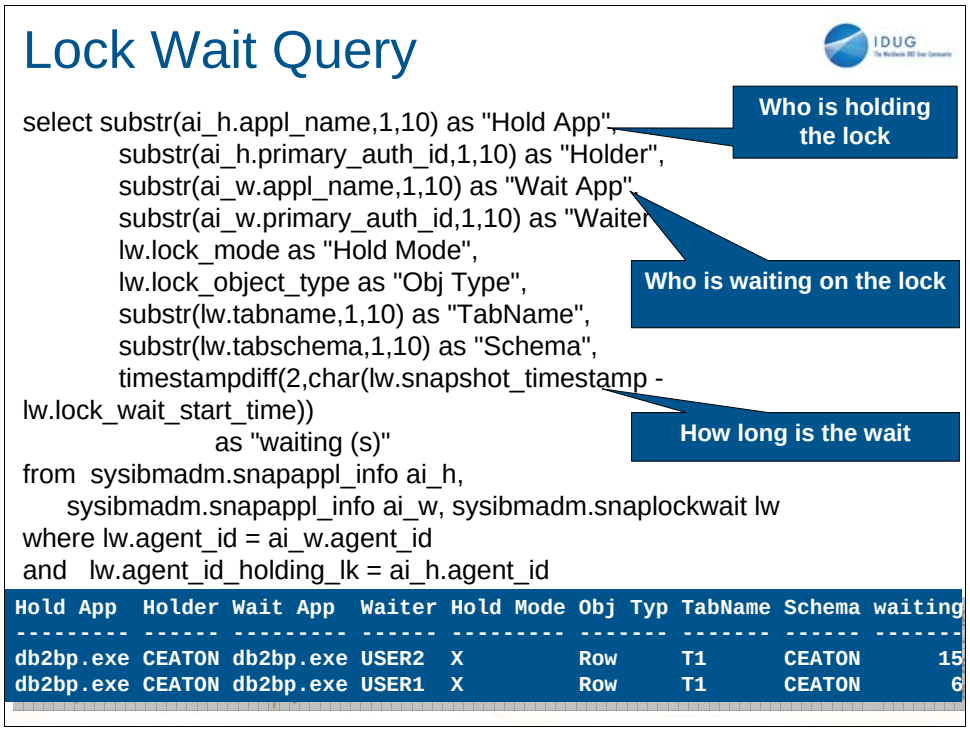
STMT	TYPE	NUM_EXECUTIONS	TOTAL_TIME (ms)	WAIT_TIME(ms)
Select * from emp	D	10	123	7
with aa as (select *	D	100	2845	860

This new function has a wealth of new information available in it and will likely be the place where more of your time will be spent going forward. The next slide also shows the additional columns that are coming soon to this monitor function.

Package Cache Query

- **Other useful bits of information in the MON_GET_PKG_CACHE_STMT function**
  - NUM_EXECUTIONS
  - PREP_TIME
  - TOTAL_ACT_TIME
  - TOTAL_ACT_WAIT_TIME
  - TOTAL_CPU_TIME
  - LOCK_WAIT_TIME
  - TOTAL_SECTION_SORT_TIME
  - TOTAL_SECTION_SORTS
  - LOCK_ESCALS
  - LOCK_WAITS
  - ROWS_MODIFIED
  - ROWS_READ
  - TOTAL_SORTS
  - SORT_OVERFLOWS
  - DEADLOCKS
  - LOCK_TIMEOUTS
  - LOG_BUFFER_WAIT_TIME
  - LOG_DISK_WAIT_TIME
  - STMT_TEXT CLOB(2MB)

Lock Wait Query




```
select substr(ai_h.appl_name,1,10) as "Hold App",
       substr(ai_h.primary_auth_id,1,10) as "Holder",
       substr(ai_w.appl_name,1,10) as "Wait App",
       substr(ai_w.primary_auth_id,1,10) as "Waiter",
       lw.lock_mode as "Hold Mode",
       lw.lock_object_type as "Obj Type",
       substr(lw.tabname,1,10) as "TabName",
       substr(lw.tabschema,1,10) as "Schema",
       timestampdiff(2,char(lw.snapshot_timestamp -
lw.lock_wait_start_time))
                as "waiting (s)"
from  sysibmadm.snapappl_info ai_h,
     sysibmadm.snapappl_info ai_w, sysibmadm.snaplockwait lw
where lw.agent_id = ai_w.agent_id
and   lw.agent_id_holding_lk = ai_h.agent_id
```

Hold App	Holder	Wait App	Waiter	Hold Mode	Obj Typ	TabName	Schema	waiting
db2bp.exe	CEATON	db2bp.exe	USER2	X	Row	T1	CEATON	15
db2bp.exe	CEATON	db2bp.exe	USER1	X	Row	T1	CEATON	6

This query shows you any lock chains that currently exist. It shows the lock holder, the application/user waiting on the lock as well as the object locked and the length of time the waiter has been waiting. It is not abnormal to see lock wait chains. What is abnormal is to see lengthy waiting times. If you see long waits, you should look at what the holding application is doing (what SQL statement and what the application status is) to determine if the application is well tuned.

```
-- Example of 2 users both waiting on a row held by CEATON
-- Hold App   Holder     Wait App   Waiter     Hold Mode Obj Type TabName    Schema     waiting (s)
-- ---------- ---------- ---------- ---------- --------- -------- ---------- ---------- -----------
-- db2bp.exe  CEATON     db2bp.exe  USER2      X         Row      T1         CEATON              15
-- db2bp.exe  CEATON     db2bp.exe  USER1      X         Row      T1         CEATON               6
```

```
-- Example of lock chain where ceaton holds X lock that user1 wants and user2 is held up behind user1
-- Hold App   Holder     Wait App   Waiter     Hold Mode Obj Type TabName    Schema     waiting (s)
-- ---------- ---------- ---------- ---------- --------- -------- ---------- ---------- -----------
-- db2bp.exe  USER1      db2bp.exe  USER2      X         Row      T1         CEATON              2
-- db2bp.exe  CEATON     db2bp.exe  USER1      X         Row      T1         CEATON             32
```

Excessive Sorting

- Show the sort time, and wait time for all sorts by connection

```
SELECT
   APPLICATION_HANDLE AS APP_HDL,
   SUBSTR(CLIENT_USERID,1,10) AS USERID,
   TOTAL_SECTION_SORTS AS NUM_SORTS,
   TOTAL_SECTION_SORT_TIME AS TOTAL_TIME,
   TOTAL_SECTION_SORT_PROC_TIME AS SORT_TIME,
   TOTAL_SECTION_SORT_TIME -
   TOTAL_SECTION_SORT_PROC_TIME AS WAIT_TIME
FROM TABLE(MON_GET_CONNECTION(NULL,-1))
```

APP_HDL	USERID	NUM_SORTS	TOTAL_TIME	SORT_TIME	WAIT_TIME
7	CEATON	36	7579	7495	84

Another new monitor function is the MON_GET_CONNECTION table function. This rolls up all the statements for a given connection and shows a wealth of information about a given connection. Here is just a sample of some of the columns (see the information center for all the details).

Column name	Data type	Description
APPLICATION_HANDLE	BIGINT	application_handle - Application handle
APPLICATION_NAME	VARCHAR(128)	Reserved for future use.
APPLICATION_ID	VARCHAR(128)	Reserved for future use.
MEMBER	SMALLINT	member- Database member
CLIENT_WRKSTNNAME	VARCHAR(255)	CURRENT CLIENT_WRKSTNNAME special register
CLIENT_ACCTNG	VARCHAR(255)	CURRENT CLIENT_ACCTNG special register
CLIENT_USERID	VARCHAR(255)	CURRENT CLIENT_USERID special register
CLIENT_APPLNAME	VARCHAR(255)	CURRENT CLIENT_APPLNAME special register
CLIENT_PID	BIGINT	Reserved for future use.
CLIENT_PRDID	VARCHAR(128)	Reserved for future use.
CLIENT_PLATFORM	VARCHAR(12)	Reserved for future use.
CLIENT_PROTOCOL	VARCHAR(10)	Reserved for future use.
SYSTEM_AUTH_ID	VARCHAR(128)	Reserved for future use.
SESSION_AUTH_ID	VARCHAR(128)	Reserved for future use.
COORD_MEMBER	SMALLINT	Reserved for future use.
CONNECTION_START_TIME	TIMESTAMP	Reserved for future use.
ACT_ABORTED_TOTAL	BIGINT	act_aborted_total - Total aborted activities
ACT_COMPLETED_TOTAL	BIGINT	act_completed_total - Total completed activities
ACT_REJECTED_TOTAL	BIGINT	act_rejected_total - Total rejected activities
AGENT_WAIT_TIME	BIGINT	agent_wait_time - Agent wait time
AGENT_WAITS_TOTAL	BIGINT	agent_waits_total - Total agent waits
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads
POOL_TEMP_DATA_L_READS	BIGINT	pool_temp_data_l_reads - Buffer pool temporary data logical reads
POOL_TEMP_INDEX_L_READS	BIGINT	pool_temp_index_l_reads - Buffer pool temporary index logical

Top Consuming Transactions

- Show the transactions with the most CPU and most Wait Time

```
SELECT
   APPLICATION_HANDLE AS APP_HDL,
   SUBSTR(CLIENT_USERID,1,10) AS USERID,
   TOTAL_RQST_TIME,
   TOTAL_CPU_TIME,
   TOTAL_WAIT_TIME,
   CLIENT_IDLE_WAIT_TIME
FROM TABLE(MON_GET_UNIT_OF_WORK(NULL,-1))
```


Similarly there is a new monitor function called MON_GET_UNIT_OF_WORK which will show you all the metrics for all the statements within a given unit of work. Additional columns include:

Column Name	Data Type	Description or corresponding monitor element
SERVICE_SUPERCLASS_NAME	VARCHAR(128)	service_superclass_name - Service superclass name
SERVICE_SUBCLASS_NAME	VARCHAR(128)	service_subclass_name - Service subclass name
SERVICE_CLASS_ID	INTEGER	service_class_id - Service class ID
MEMBER	SMALLINT	member- Database member
COORD_MEMBER	SMALLINT	coord_member - Coordinator member
APPLICATION_HANDLE	BIGINT	application_handle - Application handle
APPLICATION_ID	VARCHAR(128)	Reserved for future use.
WORKLOAD_NAME	VARCHAR(128)	workload_name - Workload name
WORKLOAD_OCCURRENCE_ID	INTEGER	workload_occurrence_id - Workload occurrence identifier. This ID does not uniquely identify the workload occurrence unless it is coupled with the coordinator member and the workload name.
UOW_ID	INTEGER	uow_id - Unit of work ID
WORKLOAD_OCCURRENCE_STATE	VARCHAR(32)	workload_occurrence_state - Workload occurrence state
CLIENT_WRKSTNNAME	VARCHAR(255)	CURRENT CLIENT_WRKSTNNAME special register
CLIENT_ACCTNG	VARCHAR(255)	CURRENT CLIENT_ACCTNG special register
CLIENT_USERID	VARCHAR(255)	CURRENT CLIENT_USERID special register
CLIENT_APPLNAME	VARCHAR(255)	CURRENT CLIENT_APPLNAME special register
UOW_START_TIME	TIMESTAMP	Reserved for future use.
SESSION_AUTH_ID	VARCHAR(128)	Reserved for future use.
ACT_ABORTED_TOTAL	BIGINT	act_aborted_total - Total aborted activities
ACT_COMPLETED_TOTAL	BIGINT	act_completed_total - Total completed activities
ACT_REJECTED_TOTAL	BIGINT	act_rejected_total - Total rejected activities
AGENT_WAIT_TIME	BIGINT	agent_wait_time - Agent wait time
AGENT_WAITS_TOTAL	BIGINT	agent_waits_total - Total agent waits
POOL_DATA_L_READS	BIGINT	pool_data_l_reads - Buffer pool data logical reads
POOL_INDEX_L_READS	BIGINT	pool_index_l_reads - Buffer pool index logical reads

Coming Soon

- Unit of Work monitor will also include

TOTAL_COMPILE_TIME	BIGINT	Reserved for future use.	
TOTAL_COMPILE_PROC_TIME	TOTAL_APP_ROLLBACKS	BIGINT	Reserved for future use.
TOTAL_COMPILATIONS	INT_ROLLBACKS	BIGINT	Reserved for future use.
TOTAL_IMPLICIT_COMPILE_TIME	TOTAL_RUNSTATS_TIME	BIGINT	Reserved for future use.
TOTAL_IMPLICIT_COMPILE_PROC_TIME	TOTAL_RUNSTATS_PROC_TIME	BIGINT	Reserved for future use.
TOTAL_IMPLICIT_COMPILATIONS	TOTAL_RUNSTATS	BIGINT	Reserved for future use.
TOTAL_SECTION_TIME	TOTAL_REORG_TIME	BIGINT	Reserved for future use.
TOTAL_SECTION_PROC_TIME	TOTAL_REORG_PROC_TIME	BIGINT	Reserved for future use.
TOTAL_APP_SECTION_EXECUTIONS	TOTAL_REORGS	BIGINT	Reserved for future use.
TOTAL_ACT_TIME	TOTAL_LOAD_TIME	BIGINT	Reserved for future use.
TOTAL_ACT_WAIT_TIME	TOTAL_LOAD_PROC_TIME	BIGINT	Reserved for future use.
ACT_RQSTS_TOTAL	TOTAL_LOADS	BIGINT	Reserved for future use.
TOTAL_ROUTINE_TIME	CAT_CACHE_INSERTS	BIGINT	Reserved for future use.
TOTAL_ROUTINE_INVOCATIONS	CAT_CACHE_LOOKUPS	BIGINT	Reserved for future use.
TOTAL_COMMIT_TIME	PKG_CACHE_INSERTS	BIGINT	Reserved for future use.
TOTAL_COMMIT_PROC_TIME	PKG_CACHE_LOOKUPS	BIGINT	Reserved for future use.
TOTAL_APP_COMMITS	THRESH_VIOLATIONS	BIGINT	Reserved for future use.
INT_COMMITS	NUM_LW_THRESH_EXCEEDED	BIGINT	Reserved for future use.
TOTAL_ROLLBACK_TIME	UOW_LOG_SPACE_USED	BIGINT	Reserved for future use.
TOTAL_ROLLBACK_PROC_TIME	ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use.

Monitoring Health And Status With SQL Select Statements

Monitoring Table Access

- Show the most active tables

```
SELECT
      SUBSTR(TABSCHEMA,1,10) AS SCHEMA,
      SUBSTR(TABNAME,1,20) AS NAME,
      TABLE_SCANS,
      ROWS_READ,
      ROWS_INSERTED,
      ROWS_DELETED
FROM TABLE(MON_GET_TABLE('','',-1))
ORDER BY ROWS_READ DESC
FETCH FIRST 5 ROWS ONLY
```

SCHEMA	NAME	TABLE_SCANS	ROWS_READ	ROWS_INSERTED	ROWS_DELETED
CEATON	WIKI_ACTIONS	14	6608	500	0
SYSIBM	SYSTABLES	16	6161	0	0
CEATON	WIKI_VISITORS	12	5664	0	70
SYSTOOLS	HMON_ATM_INFO	19	3627	0	0
SYSIBM	SYSINDEXES	0	348	0	0

There are also a set of monitor functions for objects within the database to show you how often a given object has been accessed, updated, etc.

This example shows access for table objects. Columns include

Column Name	Data Type	Description
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TABNAME	VARCHAR(128)	table_name - Table name
MEMBER	SMALLINT	member- Database member
TAB_TYPE	VARCHAR(14)	table_type - Table type. This interface returns a text identifier based on defines in sqlmon.h, and is one of:

* USER_TABLE
* DROPPED_TABLE
* TEMP_TABLE
* CATALOG_TABLE
* REORG_TABLE

TAB_FILE_ID	BIGINT	table_file_id - Table file ID
DATA_PARTITION_ID	INTEGER	data_partition_id - Data partition identifier
TBSP_ID	BIGINT	tablespace_id - Table space identification
INDEX_TBSP_ID	BIGINT	index_tbsp_id - Index table space ID
LONG_TBSP_ID	BIGINT	long_tbsp_id - Long table space ID
TABLE_SCANS	BIGINT	table_scans - Table scans
ROWS_READ	BIGINT	rows_read - Rows read
ROWS_INSERTED	BIGINT	rows_inserted - Rows inserted
ROWS_UPDATED	BIGINT	rows_updated - Rows updated
ROWS_DELETED	BIGINT	rows_deleted - Rows deleted
OVERFLOW_ACCESSES	BIGINT	overflow_accesses - Accesses to overflowed records
OVERFLOW_CREATES	BIGINT	overflow_creates - Overflow creates
PAGE_REORGS	BIGINT	Reserved for future use.
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use.

Monitoring Index Access

- Show me the indexes that have been most active
  - Metrics will only be returned for indexes on tables that have been accessed since the database was activated.

```
SELECT
    SUBSTR(TABSCHEMA,1,10) AS SCHEMA,
    SUBSTR(TABNAME,1,20) AS NAME,
    IID,   NLEAF,  NLEVELS,
    INDEX_SCANS,
    KEY_UPDATES,
    BOUNDARY_LEAF_NODE_SPLITS +
    NONBOUNDARY_LEAF_NODE_SPLITS AS PAGE_SPLITS
FROM TABLE(MON_GET_INDEX('','',-1))
ORDER BY INDEX_SCANS DESC
FETCH FIRST 5 ROWS ONLY
```

SCHEMA	NAME	IID	NLEAF	NLEVELS	INDEX_SCANS	UPDATES	SPLITS
SYSTOOLS	HMON_ATM_INFO	1	2	2	754	0	0
SYSIBM	SYSUSERAUTH	1	8	2	425	0	0
SYSIBM	SYSPLANAUTH	1	9	2	192	0	0
SYSIBM	SYSTABLES	1	6	2	186	0	0
SYSIBM	SYSINDEXES	2	5	2	145	0	0

Similar to the table monitor, there is one for indexes as well which shows all index access since the database was activated.

Column Name	Data Type	Description or corresponding monitor element
TABSCHEMA	VARCHAR(128)	table_schema - Table schema name
TABNAME	VARCHAR(128)	table_name - Table name
IID	SMALLINT	iid - Index identifier
MEMBER	SMALLINT	member- Database member
DATA_PARTITION_ID	INTEGER	data_partition_id - Data partition identifier. If index is not partitioned, NULL is returned.
NLEAF	BIGINT	nleaf - Number of leaf pages
NLEVELS	SMALLINT	nlevels - Number of index levels
INDEX_SCANS	BIGINT	index_scans - Index scans
INDEX_ONLY_SCANS	BIGINT	index_only_scans - Index-only scans
KEY_UPDATES	BIGINT	key_updates - Key updates
INCLUDE_COL_UPDATES	BIGINT	include_col_updates - Include column updates
PSEUDO_DELETES	BIGINT	pseudo_deletes - Pseudo deletes
DEL_KEYS_CLEANED	BIGINT	del_keys_cleaned - Pseudo deleted keys cleaned
ROOT_NODE_SPLITS	BIGINT	root_node_splits - Root node splits
INT_NODE_SPLITS	BIGINT	int_node_splits - Intermediate node splits
BOUNDARY_LEAF_NODE_SPLITS	BIGINT	boundary_leaf_node_splits - Boundary leaf node splits
NONBOUNDARY_LEAF_NODE_SPLITS	BIGINT	nonboundary_leaf_node_splits - Non-boundary leaf node splits
PAGE_ALLOCATIONS	BIGINT	page_allocations - Page allocations
PSEUDO_EMPTY_PAGES	BIGINT	pseudo_empty_pages - Pseudo empty pages
EMPTY_PAGES_REUSED	BIGINT	empty_pages_reused - Empty pages reused
EMPTY_PAGES_DELETED	BIGINT	empty_pages_deleted - Empty pages deleted
PAGES_MERGED	BIGINT	pages_merged - Pages merged
ADDITIONAL_DETAILS	BLOB(100K)	Reserved for future use.

SQL to View Notification Log

- Show me all the Critical and Error messages in the last 24 hours

```
SELECT TIMESTAMP, SUBSTR(MSG,1,400) AS MSG
FROM SYSIBMADM.PDLOGMSGS_LAST24HOURS
WHERE MSGSEVERITY IN ('C','E')
ORDER BY TIMESTAMP DESC
```

- Show me all the messages in the notify log from the last 3 days

```
TIMESTAMP                  MSG
-------------------------- ------------------------------------------------------------
2009-03-16-09.41.47.673002 ADM6044E  The DMS table space "SMALLTBSP" (ID "2") is
full.  If this is an autoresize or automatic storage DMS tablespace, the maximum table space
size may have been reached or the existing containers or storage paths cannot grow any
more. Additional space can be added to the table space by either adding new containers or
extending existing ones using the ALTER TABLESPACE SQL statement.
```


There is an administrative view called SYSIBMADM.PDLOGMSGS_LAST24HOURS. This view shows you the messages that exist in the notification log over the last 24 hours. The DDL for this view is as follows:

TIMESTAMP
TIMEZONE
INSTANCENAME
DBPARTITIONNUM
DBNAME
PID
PROCESSNAME
TID
APPL_ID
COMPONENT
FUNCTION
PROBE
MSGNUM
MSGTYPE
MSGSEVERITY
MSG

If you want to view notification log messages that are older then 24 hours then use the PD_GET_LOG_MSGS table function and specify the start time you want to view messages from. The columns returned from the table function are the same as those of the PDLOGMSGS_LAST24HOURS view.

SQL to View Database History

- Show the average and maximum time taken to perform full backups

```
SELECT AVG(TIMESTAMPDIFF(4,CHAR(
          TIMESTAMP(END_TIME) - TIMESTAMP(START_TIME))))  AS AVG_BTIME,
       MAX(TIMESTAMPDIFF(4,CHAR(
          TIMESTAMP(END_TIME) - TIMESTAMP(START_TIME)))) AS MAX_BTIME
```

```
START_TIME     SQLCODE     CMD
-------------- ----------- ----------------------------------
20061114093635        -204 DROP TABLESPACE IBMDB2SAMPLEXML
20061218125352       -1422 CREATE  REGULAR  TABLESPACE SMALLTSP
```

- Show any commands in the recovery history file that failed

```
AVG_BTIME   MAX_BTIME
----------- -----------
         17          25
```

IDUG 2009 Europe

30

The view SYSIBMADM.DB_HISTORY gives you SQL access to the contents of the recovery history file. You no longer need to run LIST HISTORY commands and parse the output. Instead you can run SQL scripts to look for exactly what you need from the recovery history file. The columns in this view that I think are important are (for all columns see the link below)

DBPARTITIONNUM	SMALLINT	Database partition number.
START_TIME	VARCHAR(14)	Timestamp marking the start of a logged event.
END_TIME	VARCHAR(14)	Timestamp marking the end of a logged event.
FIRSTLOG	VARCHAR(254)	Name of the earliest transaction log associated with an event.
LASTLOG	VARCHAR(254)	Name of the latest transaction log associated with an event.
BACKUP_ID	VARCHAR(24)	Backup identifier or unique table identifier.
TABSCHEMA	VARCHAR(128)	Table schema.
TABNAME	VARCHAR(128)	Table name.
CMD_TEXT	CLOB(2 M)	Data definition language associated with a logged event.
NUM_TBSPS	INTEGER	Number of table spaces associated with a logged event.
TBSPNAMES	CLOB(5 M)	Names of the table spaces associated with a logged event.
OPERATION	CHAR(1)	Operation identifier.
OPERATIONTYPE	CHAR(1)	Action identifier for an operation.
OBJECTTYPE	CHAR(1)	Identifier for the target object of an operation. The possible values are: D for full database, P for table space, and T for table.
LOCATION	VARCHAR(255)	Full path name for files, such as backup images or load input file, that are associated with logged events.
DEVICETYPE	CHAR(1)	Identifier for the device type associated with a logged event. This field determines how the LOCATION field is interpreted. The possible values are: A for TSM, C for client, D for disk, K for diskette, L for local, N (generated internally by DB2), O for other (for other vendor device support), P for pipe, Q for cursor, R for remote fetch data, S for server, T for tape, U for user exit, and X for X/Open XBSA interface.
SQLCODE	INTEGER	SQL return code, as it appears in the SQLCODE field of the SQLCA.
SQLSTATE	VARCHAR(5)	A return code that indicates the outcome of the most recently executed SQL statement, as it appears in the SQLSTATE field of the SQLCA.

Operation values and their associated types can be found here
http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0022351.htm

Finding the Log Hog

- Display information about the application that currently has the oldest uncommitted unit of work

```
SELECT    AI.APPL_STATUS as Status,
          SUBSTR(AI.PRIMARY_AUTH_ID,1,10) AS "Authid",
          SUBSTR(AI.APPL_NAME,1,15) AS "Appl Name",
          INT(AP.UOW_LOG_SPACE_USED/1024/1024)
               AS "Log Used (M)",
          INT(AP.APPL_IDLE_TIME/60) AS "Idle for (min)",
          AP.APPL_CON_TIME AS "Connected Since"
FROM      SYSIBMADM.SNAPDB DB,
          SYSIBMADM.SNAPAPPL  AP,
          SYSIBMADM.SNAPAPPL_INFO AI
WHERE     AI.AGENT_ID = DB.APPL_ID_OLDEST_XACT
  AND     AI.AGENT_ID = AP.AGENT_ID;
```


In this example we are looking for the information about the application that is currently the oldest transaction that is holding up the log tail. This transaction represents the total amount of recovery log that must now be scanned in order to perform crash recover. That is we start from the log file with the oldest uncommitted unit of work and read through the log files to the end in order to perform redo and undo recovery in the event of a failure. If you are running out of active log space, it may be because an uncommitted transaction has been sitting there for a long time (maybe someone went out for coffee).

There is a LOT of information in these snapshot tables. Too much to go through in this one slide. So I will leave it to you to read through the documentation on these administrative views. I'm sure you will find useful nuggets of information you can use.

SNAPAPPL_INFO

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0021987.htm

SNAPAPPL

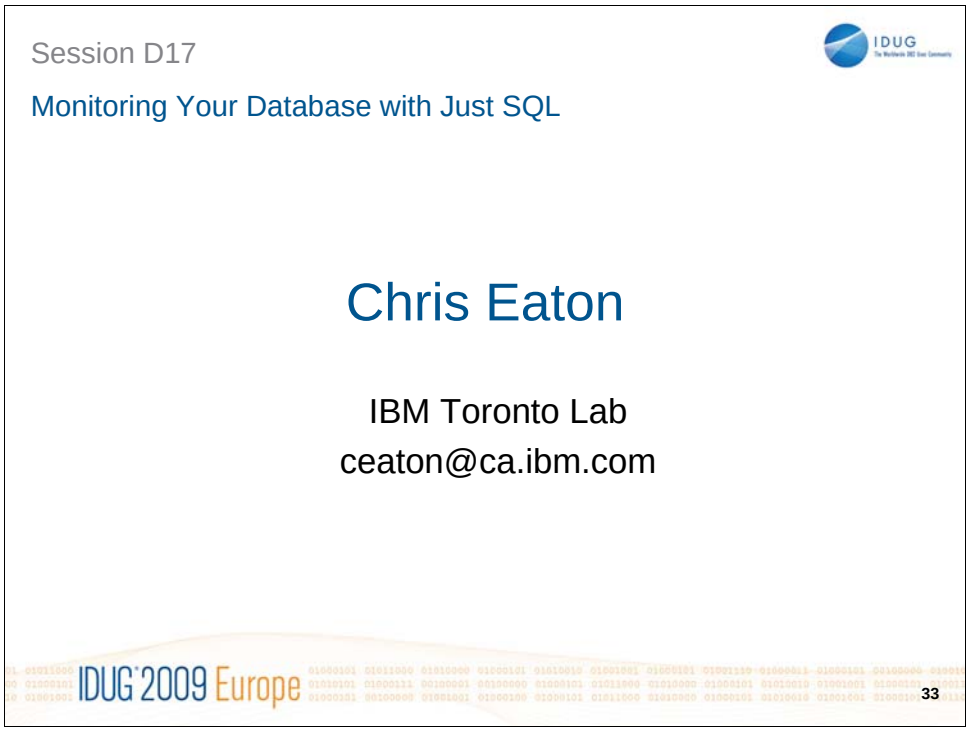
http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0021986.htm

SNAPDB

http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0022003.htm

Summary

- Monitoring in DB2 is changing rapidly
  - Moving to time spent and time waiting metrics
  - Each release and fixpack will be adding more monitor elements you can track
- Much of the support is targeted at helping tool vendors
  - However, you can use SQL to get at the same info



Chris Eaton is Senior Product Manager for DB2 primarily focused on planning and strategy for DB2. Chris has been working with DB2 on the Linux, UNIX, Windows platform for over 16 years. From customer support to development manager, to Externals Architect and now as Product Manager for DB2, Chris has spent his career listening to customers and working to make DB2 a better product. Chris is the author of “IBM DB2 9 New Features” and “The High Availability Guide for DB2” and has one of the most popular blogs about DB2 on IT Toolbox at http://it.toolbox.com/blogs/db2luw