DIAG4DB2© – The Complete DB2® App Diagnostic Solution

Do you develop, test, maintain, support, sustain, operate, secure or audit software applications that access IBM DB2® databases? If so, there's a tool named DIAG4DB2© that can help you with your Problem Determination efforts.

When asked to name the single most important toolset to our productivity, most of us answer that to be the diagnostic tools we use to help us find and resolve problems within our code. Bug and performance Problem Determination is a large personnel, cost and time consumer for most apps, particularly once they are deployed. DB2® has excellent DML and procedural language implementations, as well as a variety of database-centric diagnostics. However, its application-centric diagnostic facilities fall short for I/T professionals looking for an easy-to-use, comprehensive, production-capable diagnostic solution.

At Terallel Systems, we decided there must be an answer. And that led us to build the Diagnostic Toolkit for DB2 apps, aka DIAG4DB2©.

Highlights

DIAG4DB2© breaks through the walls of conventional basic tracing. It leverages the incredible power of SQL to both collect and report trace data. It provides what technicians need for fast, effective Problem Determination:

- Fine-granularity setup options let you define exactly what to trace – and what -not- to trace. Unlike many other diagnostic tools, DIAG4DB2© does not attempt to guess at what is problematic and what is not. Rather, it allows you (the application expert) to define that. Some of the many options, definable in any combination, can include specific:
    - Proc/function/trigger name(s)
    - Loop/block name(s) within routine
    - Unexpected loop iteration count (you define the expected iteration range)
    - Statement name(s) within routine
    - Type(s) of SQL statement (eg: SELECT, DELETE, etc)
    - Long-running duration (you define the elapsed time threshold)
    - Short-running duration (you define the elapsed time threshold)
    - Timestamp date/time start-end time range(s)
    - User(s)
    - Of-interest key value(s) or amount value
    - SQL condition(s), error code(s) or error state(s)
    - Unexpected SQL result row count (you define the expected row count range).

The scope of diagnosis is easily expanded or reduced via handy options such as:

- Comma-delimited concatenations of multiple names
- Inclusion vs Exclusion flags
- Flags to auto-include subordinate procs, functions and/or triggers
- Flags to mark each component as active vs inactive, which allows you to specify the component once then turn it on/off at will
- Partial-key Varchar values with associated match operators
- Boolean match operators for numeric key values.

No proprietary interface is needed to change these settings; just use your familiar IDE to specify what you want.

- Full SQL SELECT column-subset and row-search power against all trace data. Run your trace queries from any of your IDEs – they're just normal SELECT statements. No special proprietary tool is needed to start a trace; no arcane location/format is needed for trace viewing. And no looking through gobs of trace mish-mash to find the one nugget you really need. Here's a simple SELECT result showing proc, block/loop and statement run times within proc TEST_DIAG_MGR within a specified session:

	A	C	D	P	AF	AG	AI	AJ	AK	AQ	AR	AS	AY
1	DBMS_SESS_ID	DIAG_CLASS_ID	PROGRESS_ID	TRACE_CTR	ROUTINE_ID	ROUTINE_SUBCLASS_ID	ROUTINE_EXEC_SECS	BLOCK_ID	BLOCK_ITER_CTR	BLOCK_EXEC_SECS	STMT_ID	STMT_OPER_ID	STMT_EXEC_SECS
2	127.0.0.1.63353.191120032052	ROUTINE	START	1	TEST_DIAG_MGR	PROC							
3	127.0.0.1.63353.191120032052	BLOCK	START	2	TEST_DIAG_MGR	PROC		TEST_BLOCK					
4	127.0.0.1.63353.191120032052	STMT	START	3	TEST_DIAG_MGR	PROC		TEST_BLOCK	1		TEST_STMT	UPDATE	
5	127.0.0.1.63353.191120032052	STMT	END	4	TEST_DIAG_MGR	PROC		TEST_BLOCK	1		TEST_STMT	UPDATE	0.106
6	127.0.0.1.63353.191120032052	STMT	START	5	TEST_DIAG_MGR	PROC		TEST_BLOCK	2		TEST_STMT	UPDATE	
7	127.0.0.1.63353.191120032052	STMT	END	6	TEST_DIAG_MGR	PROC		TEST_BLOCK	2		TEST_STMT	UPDATE	0.044
8	127.0.0.1.63353.191120032052	BLOCK	END	7	TEST_DIAG_MGR	PROC		TEST_BLOCK	2	0.159			
9	127.0.0.1.63353.191120032052	ROUTINE	END	8	TEST_DIAG_MGR	PROC	0.173						

- NO need to attempt customer problem reproduction in a vendor or development testbed; NO need to get a copy of the customer's data and schema (which may violate privacy laws); NO need for 'debug mode' tools that require recompiling – which is often prohibited after deployment/installation. Trace right on deployed PRODUCTION systems, by briefly activating a trace customized for the specific problem.

- Open Portability. DIAG4DB2© can be used to trace your application execution progress and its SQL issued from all sorts of programming languages – from SQL procedural language inside the DB2® server, to Java, C, COBOL or any other language that allows an SQL CALL statement to be executed from any client-side (and/or application-server-side) environment. It's a single, simple, universal solution – learn it once in a few minutes and use it across all of your languages, IDEs and platforms.

- Turns out, DIAG4DB2© can also perform many other valuable diagnostic/debugging actions BEYOND trace, all with the same rich granularity options:
    - Display a customized real-time monitor of selected DB2® sessions with detailed progress statistics and color-coded alerts. No clutter/confusion typical of an all-session displays, just the sessions YOU want to see. Options for cell-level color-coding to build in visual alerting upon your supplied thresholds. Accessible on any client platform that can view a Microsoft Excel spreadsheet! Here's a sample:

	A	B	C	J	K	M	P	Q	S	W	X	Z
1	DBMS_SESS_ID	DBMS_USR_ID	APP_ID	MR_MONITOR_ROUTINE_ID	MR_MONITOR_ROUTINE_CURRENT_ELAPSED_SECS	MR_MONITOR_ROUTINE_ALERT_COLOR_ID	MR_MONITOR_BLOCK_BLOCK_ID	MR_MONITOR_BLOCK_CURRENT_ELAPSED_SE	MR_MONITOR_BLOCK_ALERT_COLOR_ID	MR_MONITOR_STMT_STMT_I	MR_MONITOR_STMT_CURRENT_ELAPSED_SE	MR_MONITOR_STMT_ALERT_COLOR_ID
2	127.0.0.1.50646.190927020315	DANH	A/R	STND_PAYMENT						UPD_CUST_MSTR	13	GREEN
3	127.0.0.1.50661.190927020705	KENC	ORDER	SALES_RPTING			ORDR_MSTR_LOOP	7	GREEN	REGN_RPT		
4	127.0.0.1.50692.190927030334	SHARONT	BILLING	INVOICE_GEN	4	GREEN						
5	127.0.0.1.50702.190927033520	LEMW	ORDER	SALES_RPTING			ORDR_MSTR_LOOP	2	GREEN	REGN_RPT		
6	127.0.0.1.50735.190927040219	ALANC	I/T	SCHEMA_MAINT						CR_TEMP36	41	GREEN
7	127.0.0.1.50811.190927053920	TODDW		DBBACKUPS	83	GREEN				BU_TESTDB		
8	127.0.0.1.50902.190927054453	JACKS	FINANCE	CORP_MAINT	37	RED				ADD_INVESTOR	6	YELLOW
9	127.0.0.1.51041.190927062021	PHILN		BI_TOOL						CORP_RPT	2	GREEN
10	127.0.0.1.51277.190927062850	SUSANW	PAYROLL	WEEKLY_PAY			EMPL_PAY_LOOP	12	YELLOW	CALC_PAY	11	RED
11	127.0.0.1.51410.190927070302	CHUCKM	I/T	PERIODIC_CLEANUP						PURGE_TEMP3	1	GREEN
12	127.0.0.1.51772.190927073102	KARENH		REFRESH_STATS	42	GREEN				LOAD_BILL_STATS		
13	127.0.0.1.51926.190927075341	STEPHANC	ENGR	DRAWINGS_MAINT			PROD_MSTR_LOOP	161	RED	UPD_PROD	1	GREEN

    - Trigger automatic email notifications, eg: upon encountering unexpected errors, selected under-investigation key values and/or amounts, past-expected elapsed times, etc. Recipient(s) can be specified, and/or auto-routed to the running end user.

    - EXPLAIN an SQL statement for performance investigation, permitting detailed path analysis and/or visual path depiction.
    - Pause execution at any point, permitting you to control execution progress while investigating trace, monitor, Explain and/or application database tables' content.

    - Signal a user-defined SQL condition, which the application can then check for and halt via a simple handler.
    - Even run custom DB2® procs that you can write yourself.

What started out as a powerful tracer tool has grown up into a mature, comprehensive, multi-action diagnostic toolkit. Production app debugging just got easier.

Feature Details

Us techies want more tech detail, right? Well here's just a few of the DIAG4DB2© feature details to wet your whistle:

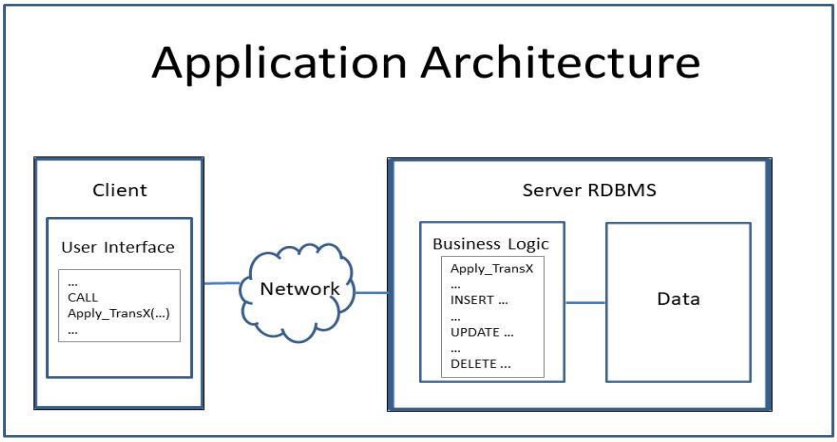
- Complete audit tagging of each trace row with:
  - Exact timestamp
  - Unique source object identifier
  - Database session identifier and execution sequence counter within session
  - Exact type of SQL statement or database command
  - Full end-to-end integrated, easily-readable execution stack, including BOTH client-side AND server-side (SQL PL) application logic
  - Optional app-specific key values
  - A tidy summary set of performance metrics (CPU, I/O, locking, networking)
  - Much more detail, with inclusion/exclusion options for potentially-large and/or secure fields.
- Diagnostic trace data rows DO NOT EVAPORATE when your app does a ROLLBACK or your app session ends. They are managed and committed via processes totally independent of the user app.
- Auto-calculated routine, block/loop and statement execution time durations – which are then of course columns you can query in your SELECTs against the trace data. For example, find the 10 slowest-running SQL statements in your app. Or find the daily counts of times that problem-statement/loop/routine XYZ took longer than 2.5 seconds to run. Easy Peasy with SQL SELECT.
- Optional capture of static and dynamic SQL statement text content – because sometimes you've just got to see the gory detail.
- Ability to capture specific point-reached with variables' current values.
- Dynamically activate/deactivate/modify any diagnostic setting(s) WHILE your app is running. DIAG4DB2© rapidly recognizes these changes and adjusts accordingly.
- NO artificial limits on diagnostic content size or retention duration. Selectively delete diagnostic rows whenever you decide – via simple SQL DELETE statements, of course. And before you run those DELETEs, you can optionally save any interesting subset of those rows to personal/project/archival permanent tables via simple INSERT-SELECTs.

- Ability to capture all SQL result diagnostic detail (result row counts, messages, intra-message tokens, etc.), not just error codes.
- Ability to choose which loop iteration(s) should participate. For example, include only iterations 1-2, include only every 100th iteration, etc.
- User-defined action maximums, to protect against excessive volume for cases such as large/infinite loops, UDFs within SQL statements which reference high row-count tables, etc.
- Supports per-user personal diagnostic profiles. Multiple users can run different diagnostics on the same app concurrently – no conflicts, no queues.
- Trace and monitor tables show REALTIME app execution progress – no deferred/delayed post-processing of RDBMS logs.
- Ability to trigger actions immediately upon passing of an expected maximum elapsed time. For example, you could be automatically notified of a routine, block/loop or statement that is currently running longer than expected. You could begin investigation by examining automatically generated Trace and/or Explain rows, nipping a problem in the bud instead of waiting for execution to complete hours later with a resulting fiasco.
- Ability to specify elapsed time thresholds of very small durations, eg: 20 milliseconds. Over three orders of magnitude smaller than DB2®'s threshold limit!
- Ability to reduce lines of code within your app through use of advanced options:
    - Expanding the basic SQL error-handling constructs to include a wider set of errors, such as unexpected result row counts and unexpected loop iterations
    - Use of DIAG4DB2© Calls that automatically perform both dynamic SQL execution as well as before/after diagnostic checking
    - Use of application condition setting to announce application-side errors. Handling of each such error can then be custom configured, such as creation of a trace row followed by a termination signal.
- Efficient, low-overhead diagnostic checking and action execution, even when used comprehensively. DIAG4DB2© knows how to:
    - Leverage DB2®'s high-speed built-in constructs
    - Offload work to asynchronous, out-of-application processes
    - Check and optimize for conditions like all action limits achieved.

  The result is often near-zero application performance impact, even for high-volume OLTP apps.

- Allows per-install rule settings when errors are encountered. For example, your app can terminate on data quality errors for some installs, but continue on for installs where the customer is willing to proceed.
- User options do not open exposure to RCE risks, unlike other tools (eg: the Log4j disaster). Diagnostics are performed inside the security of the database manager, not out in the wild of the host. Dynamically formed SQL is checked with built-in bulletproofing against malicious user SQL insertion attempts. Even tool installation contains no host-side executable, just safe SQL scripts.

- The entire SQL stored proc/UDF/trigger programming environment becomes much more capable of supporting complex application logic, since necessary diagnostic options can be built into the logic. This allows app designers to move more business logic out of the client and into the RDBMS, gaining considerable improvements in performance, security, reliability, change administration/coordination, network traffic volume, etc. Existing apps can relocate logic; new apps can position logic in the RDBMS from the get-go. The longstanding RDBMS promise of tightly integrating business logic with data can finally be realized!



- OK, so DIAG4DB2© helps solve technical problems like unexpected error codes, broken code logic and poor performance, right? Yes, but DIAG4DB2© doesn't stop there. It can expand the entire scope of 'problem' coverage to include issues managed by security analysts, financial auditors, fraud analysts, etc. Business problems that most traditional diagnostic tools have no clue how to address. Now that's a pretty big claim. You're probably wondering how a diagnostic tool could possibly do all that too. Well, in addition to its flexible predefined filtering, DIAG4DB2© also allows you to perform custom filtering via your own SQL procs that can do things like lookups into your own custom watch list SQL tables. Things like user ID watch lists, IP address watch lists, account number watch lists, transaction amount watch thresholds, etc. Ah, light bulb now on!

Usage

DIAG4DB2© uses a source code in-line framework – you add SQL CALLs in your application source code to invoke diagnostic checking wherever you desire.

These calls are simple to code, and they check for and perform any applicable currently-active diagnostic actions. Meanwhile, you control all definitions, authorizations and activations/deactivations by maintaining rows in a simple set of DIAG4DB2© database tables. Presto – a dynamically-controlled, custom diagnostic action suite is yours!

Some common usage approaches include:

- In SQL condition handlers, such that all or selected warning/exceptions are always checked
- In client-side routines and/or server-side SQL procs, SQL user-defined functions and/or SQL triggers that have recently or historically caused functional error or performance problems. You can further focus on particularly problematic DO/WHILE/UNTIL/FOR loops and/or specific SQL statements within these routines. Use the 80/20 (or 90/10 or 99/1) rule to your advantage – beef up diagnostics in the small fraction of your code that causes the majority of your problems.
- In centralized 'data access' routines that are called by other routines within your app.
- Around SQL that needs a full permanent audit trail (for example, session startup and end)
- Around SQL that may need special security handling (for example, trace and notify a security analyst whenever a transaction is executed by a user ID in their watch list)
- Around SQL that may need special audit (for example, trace and notify a financial auditor whenever a transaction uses an excessive currency amount)
- Around SQL that may be using suspicious key values (for example, trace and notify a fraud analyst whenever a transaction uses an account ID in their watch list)

- Comprehensively throughout the app, for maximum diagnostic flexibility.

Summary

Software development organizations large and small are constantly searching for ways to increase their install base WITHOUT increasing their support costs. DIAG4DB2© helps achieve this critical objective.

When the fix pressure is on, let DIAG4DB2© help you find those bugs fast, and:

- Boost your Developer, Tester, Customer Support, Call Center and CSR productivity
- Expand your Problem Determination capability
- Reduce your problem incident resolution times
- Meet and exceed your customer support Service Level Agreements
- Reduce your internal problem-replication sandbox infrastructure costs
- Improve your customer satisfaction and retention, via the rapid fixes you provide them.

Also don't forget about your Operations, Audit, Security and Compliance associates. You can use DIAG4DB2© as well to improve their productivity and expand their watchdog abilities.

To find out more, contact us at:

sales@terallel.com

513-827-6932