

What This Course Is

- **Intensive three-day Ingres Developer course**
 - Focus is on C and ABF programming, but this is not a “how to” for those languages, but rather a “how best” that also applies to other languages
- **Introduces students to the Ingres Relational Database Management System**
- **Assumes some existing Ingres experience in order to receive the maximum benefit from this course**
- **Teaches students methods to develop applications that perform well and are easier to maintain**
- **Reviews best practices**



www.Comp-Soln.com
©1999-2006 Comprehensive Consulting Solutions, Inc. All rights reserved.

Anticipated Results

- **Gain an understanding of improved coding techniques, and application design for performance and to minimize the occurrence of problems**
 - **Makes for systems that are easier to maintain and have fewer data integrity and concurrency problems**
- **Gain a working knowledge of Ingres architecture - something that will help a Developer achieve optimum results**
- **Gain an understanding of good transaction and query design**
- **Gain an understanding of what is typically expected from a professional developer**
- **Understand best practices that will help you become a better Developer!**



www.Comp-Soln.com
©1999-2006 Comprehensive Consulting Solutions, Inc. All rights reserved.

Making Changes

- **Define a rigid change management process**
- **Perform an Impact Analysis of the change**
 - A simple change, such as renaming a column or changing the attributes (e.g., character to integer) could result in thousands of changes in hundreds of programs
- **Document all changes**
 - This makes it easy to recreate
 - Easy to back out if necessary
- **Maintain Version Control & Build Manifest**
 - You should be able to re-build any version of a working application!



www.Comp-Soln.com
©1999-2006 Comprehensive Consulting Solutions, Inc. All rights reserved.

Making Changes – continued

- **Benchmark Before and After**
 - Have a clear understanding of performance and/or concurrency requirements. It is far better to design performance and concurrency into a system than to try to “back it in” later.
 - Never assume anything! Please note that this statement does not just apply to benchmarking!
 - Validate data integrity and transaction design with any change
 - It is not good practice to sacrifice performance and/or concurrency in order to implement new functionality



www.Comp-Soln.com
©1999-2006 Comprehensive Consulting Solutions, Inc. All rights reserved.

ACID Properties

- **A - Atomicity:** Every transaction is discrete and atomic
- **C - Consistency:** Transactions move from one consistent state to another (never ambiguous.)
- **I - Isolation:** Each transaction behaves like it is the only transaction in the system
- **D - Durability:** Once committed a transaction is durable
- I.e., you cannot roll back a commit

- **NOTE: The use of “readlock = nolock” violates the ACID properties!**



www.Comp-Soln.com
©1999-2006 Comprehensive Consulting Solutions, Inc. All rights reserved.

Transactions - SQL

- **Definition:** A logical unit of *recoverable* work
- May be divided into multiple physical units of work
- One or more SQL statements that are processed as a single action (a multi-query transaction, or MQT)
- Starts automatically when you issue an SQL statement
- Ended by a “rollback” or a “commit” (implicit or explicit)
- It is possible to “set autocommit on” to have every statement automatically committed
 - **Not recommended since it can introduce data integrity problems**
- Results of a transaction are only visible to other users upon a commit (I.e., serializability)



www.Comp-Soln.com
©1999-2006 Comprehensive Consulting Solutions, Inc. All rights reserved.

Best Practices – Transaction Design

- **A logical business transaction may be broken into several business transactions**
 - **Need to balance concurrency with data integrity**
- **Recoverability and data integrity are key**
- **Consistent access time is another key goal**
 - **Use fixed rather than conditional logic for navigation within the transaction**
 - Variable execution paths create differences in runtime performance and can be difficult to debug
 - Discrete functions are better than code that is skipped 80% of the time
 - **Source code functions should have one entry point and (ideally) one exit point**



www.Comp-Soln.com
©1999-2006 Comprehensive Consulting Solutions, Inc. All rights reserved.

Best Practices – Touch Data Only Once

- **OHIO - Only Handle It Once**
 - **Additional handling is wasted effort and seldom adds value to the process**
- **Whenever possible, only select the same piece of data once**
 - **Keep previously accessed static data in an array or linked list**
 - **Check the list, if the item is not found then select it from the DB and load it into the array or list**
- **Try to update a single row once per transaction, rather than multiple times**
 - **This is beneficial from a logging perspective as well**



www.Comp-Soln.com
©1999-2006 Comprehensive Consulting Solutions, Inc. All rights reserved.

Life of a Query

- Query arrives in the server.
- Stage #1:
 - Incoming text is stored in the QSF
 - Parser (PSF) is used to parse the query
 - System catalog (permits, etc.) information is requested from the RDF by the PSF
 - Query tree is generated and stored in the QSF cache (pool)
- Stage #2:
 - Optimizer (OPF) takes query tree and develops a query execution plan (QEP) using stored statistics
 - QEP is stored in the QSF cache



www.Comp-Soln.com
©1999-2006 Comprehensive Consulting Solutions, Inc. All rights reserved.

Life of a Query - Continued

- Stage #3:
 - The query execution facility (QEF) executes the QEP
 - All data access is performed by the data manipulation facility (DMF)
- Stage #4:
 - Control of the session is returned to the SCF
 - Data returned to the client. (either locally or over Ingres/Net)

Performance Note: When "repeated" queries are used Step #2 is skipped after the first execution of that query (assuming that the QSF Pool is large enough to keep the query plan cached). Often a large part of runtime is related to optimization, so it is important to understand this process



www.Comp-Soln.com
©1999-2006 Comprehensive Consulting Solutions, Inc. All rights reserved.

VIFRED – the Visual Forms Editor

- **Forms are stored in an Ingres database**
 - They can be stored in *any* database
 - This can potentially create problems when there is not a consistent approach to forms storage, or if there is missing documentation regarding where the components reside to rebuild the application.
- **Forms may be modified for various reasons:**
 - Size or placement on the screen
 - Display attributes (e.g., reverse video, underline)
 - Tab ordering

Note: It is OK to modify the forms that were generated by Vision (e.g., to improve the appearance)



www.Comp-Soln.com
©1999-2006 Comprehensive Consulting Solutions, Inc. All rights reserved.

VIFRED – the Visual Forms Editor

- **Forms may be compiled into the application to improve performance**
 - This is a recommended best practice
- **Forms may run in several modes (e.g., display or update)**
- **The mode is a function of how the form is used, and not of the form itself**
- **Forms can exceed the “screen size”, but this is considered bad practice**
- **Table fields that provide scrolling regions are available**



www.Comp-Soln.com
©1999-2006 Comprehensive Consulting Solutions, Inc. All rights reserved.

- Performing a “select * into :structure” is usually not advisable. This needlessly ties the code to a particular table definition. If you must do this, make sure that it is well documented since any change to the referenced table will require a recompilation of your program.
 - The ANSI standard is to select each individual column into a variable
 - While selecting everything into a structure is easier, it can also have negative performance implications since it will always force a retrieval from the base table (even if an index will satisfy the request)
- Never insert or update a table using a structure. This is nonstandard and can introduce problems.

- Test should be both reasonable and complete. When in doubt make the test more comprehensive.
- The testing environment should be stable enough to allow you to reproduce and bug or action at will
- A detailed log (input, desired action, actual action) will assist in debugging any problems encountered
- Any debugging code should be within an “#ifdef” so that the program can be compiled with or without the code based on a “#define”
 - Use of environment variables to enable and disable tracing / debugging “on the fly” is also acceptable but may slightly increase the workload of the application