“When you’re busy cutting, shaping, and polishing the gems (Sakai Modules), it is difficult to keep the ring setting (enterprise infrastructure) in mind.”

Indiana University has learned a few simple things that can help reduce the amount of re-cutting and/or re-shaping of Sakai gems to fit the Sakai ring.

Read on. It may save you resources, time and money in the future.
"When you're busy cutting, shaping and polishing the gems, it is difficult to keep the ring setting in mind." -- Anonymous

If we were to view Sakai modules like Gradebook, Message Forum and others as the gems, and the infrastructure supporting them as the ring setting, then this analogy seems to fit the development efforts we have experienced at Indiana University.

At Indiana University, Sakai development seems to face two enterprise infrastructure challenges:

1) Integration of data into our enterprise systems and processes
2) A broader vision of the long term use and maintenance of each Sakai module that goes beyond its primary function. This would include planning for archive/purge processes, performance, scalability and data recovery due to user error, etc.

It is our perception, that many institutions will be (or are currently) faced with needs similar to those identified above.

While the enterprise infrastructure can and should adapt to current and future needs, the reality is, in some instances, that it will be much easier to shape the modules (gems) to fit the enterprise infrastructure (ring). Just like a jeweler has to cut and shape the gems with the ring in mind, Sakai would profit by keeping a few fundamentals in mind during the development process. If this is not done, then re-cutting and/or re-shaping of the Sakai Module (gem) will most likely be necessary.

Like every jeweler’s ring will be different, it is recognized that each Institution’s Enterprise Infrastructure will be different and it is impossible to foresee all future and current needs of the user community. However, we believe that there are a few things the development community can do to initially shape (and minimize the need to re-shape) the modules (the gems) to accommodate the Enterprise Infrastructure (the ring) needs identified above. Hopefully, this can be accomplished without significant negative impact to the development process.

These would include:

1) The integration of Specialized Administrative Data designed to accommodate enterprise integration, partitioning, purge and archive, report writing, etc.
2) The Avoidance of Parsing of data in columns to obtain critical data for the operation of the Sakai application.
3) The Use of Human Recognizable Names in the database

Addition of Administrative and Enterprise-Oriented Data Columns

Indiana University, who has had Sakai in Production since July 2005, has a pressing need to archive older Sakai data and purge from the underlying Sakai database.

We anticipate that few, if any, Sakai Institutions will want to keep all data for all time in its primary database. Most will face this same need to archive older data or at least purge older data to maintain system performance, reduce computing resources and/or comply with institutional records retention policies.

After discussion with IU’s Sakai Development Team, it appears that performing archive and purge activities on the database in its current state, while not impossible, would be a rather large effort that would unfortunately, detract notably from other pressing developmental priorities. Therefore, we set out to find ways to do this in an ongoing manner that once set up would be easily maintainable. No doubt, there are several ways to do this. The trick is finding a method that lets us fit our Sakai gems (modules) into our Sakai ring (enterprise infrastructure) setting without substantially re-cutting and re-shaping the modules.

Throughout this effort, there is one thing that we have clearly concluded. “It is better to plan and integrate archive / purge (and partitioning) capabilities during the original development cycle than it is to attempt to retrofit it into existing Sakai Modules where archive and purge were not given a great deal of thought during the original development cycle.”

To accomplish archive, purge, data partitioning, data recovery due to user error and/or data capture for auditing purposes, we identified three basic approaches—each of which would require the addition of a set of specialized administrative columns designed to ease the identification of rows by time, user and activity.

This set of Specialized Administrative Columns could consist of some combination of the following:

- create_date – the date the record was first created
- mod_date – the date the record was last modified
- mod_user – the actual user identifier of the individual who performed the latest modification
- mod_type – the type of modification made – insert, update, deactivate/delete. (Deletion of deactivated records would occur periodically)
Sakai Lessons Learned at Indiana University

Sakai: Building for the Future Now

• mod_active – ids the record as active or inactive

These additional specialized administrative columns are aimed at providing easier, faster and more accurate:

• archiving, purging, and partitioning of Sakai data (create_date, mod_date, mod_active)
• extraction and integration of Sakai data into the enterprise and departmental systems (create_date, mod_date, mod_active)
• analysis and report writing (create_date, mod_date, mod_active)
• data recovery for data loss due to deletions done prematurely or in error (create_date, mod_date, mod_user, mod_type, mod_active)

Indiana University has derived three potential approaches for Sakai developers to use the Specialized Administrative Columns while developing their Sakai gem(s):

• The Early in Development Cycle Approach
• The Auditing Approach
• The Hybrid Approach

Two of the three approaches that are described throughout this document, would use shadow tables. Shadow tables would be identical in structure to their corresponding Sakai table but without indexes, constraints and/or keys. These tables would hold copies of rows that have been modified when used for auditing purposes or only deleted rows when used to protect the users from accidental data loss. In other words, they "shadow" the data in Sakai tables prior to its modification in the events of update and delete and provide an identical copy in the event of an insert. Shadow tables are intended to hold modified data for a short period of time only. For example, Indiana University deletes data that is older than 31 days from the shadow tables.

This document will distinguish the type of table being discussed by referencing them as Sakai tables when the data are supplied by the user and/or application and shadow tables when the data are captured rows stored for auditing / data recovery.

The Early in Development Cycle Approach would consist of adding the Specialized Administrative Columns into each Sakai table. The application code would populate these fields. If there were a desire to be able to easily recover data deleted prematurely or in error, the application code could be written to deactivate the rows (parent and child) instead of performing a delete. If this option were to be implemented, it would require the application code to reference the mod_active field when making SQL select calls to exclude/include deactivated rows depending upon the type of SQL action being called.

It is named as such because it would most likely appeal to developers who are in the early stages of developing a Sakai gem.

Its main advantages would be that it would:

• be fully integrated into the design process – not retrofitted
• provide the necessary columns to more easily facilitate archive, purge, partitioning, auditing and/or recovery of deleted data
• not (necessarily) require the use of triggers
• not require the use of shadow tables as required by the other approaches
• probably (of the three approaches) impact user response time the least
• make recovery of data lost due to user error much easier since rows could be deactivated and not actually deleted until after a specified period

Its main disadvantages would be that:

• it would require the addition of application code to populate the specialized columns for every data manipulation language call
• it would require the addition of application code to check to see if the row is active or not for SQL select calls.
• it would require specialized application code to mark parent and child records as deactivated which means additional code beyond what the typical delete process may be needed.

The Auditing Approach – like the Early in Development Approach, this approach would consist of adding a subset of the specialized administrative columns into each Sakai table. Triggers would be used to populate the specialized administrative fields in the Sakai tables. In addition, triggers would copy the rows in the Sakai tables into its associated shadow table. In the event of update or delete, the copy of the Sakai table row into its corresponding shadow table would occur prior to the modification to the Sakai table row. Upon insert into the Sakai table, an identical insert would be made into the shadow table.

Its main advantages would be that it would:

• NOT require modification of the existing application code
• facilitate easier, faster and more accurate analysis and report writing
• allow the retrofitting of the desired specialized administrative columns needed for archive, purge, partitioning
• make recovery of data lost due to use error much easier since deleted rows could be restored from the shadow tables
• provide a complete record of activity for a defined period of time – all inserts updates and deletes performed against the specified Sakai table

Its main disadvantages would be that it would:

• require the addition of the specialized administrative columns to the Sakai tables:
• require the addition of a shadow table for each Sakai table
• require the addition of triggers to populate the specialized columns for every data manipulation language call for the Sakai tables
Sakai: Building for the Future Now

- require the addition of triggers to the Sakai tables to copy rows to the corresponding shadow table prior to its modification in the Sakai table
- require the development, maintenance and periodic use of shadow table cleanup routines.

The Hybrid Approach would be a combination of the approaches 1 and 2. This approach would consist of creating a shadow table as a copy of each Sakai table to hold only deleted records from Sakai tables. Like the first two approaches, the Hybrid approach would require the addition of the specialized administrative columns. It differs from the Auditing approach in that in it the shadow tables would not contain copies of rows from the Sakai tables that were inserted or updated.

Its main advantages would be that it would:

- NOT require modification of the existing application code
- allow the retrofitting of the desired specialized administrative columns needed for archive, purge, partitioning
- facilitate easier, faster and more accurate analysis and report writing
- make recovery of data lost due to user error much easier since deleted rows could be restored from the shadow tables

Its main disadvantages would be that it would:

- require the addition of the specialized administrative columns to the Sakai tables:
- (optionally) require the addition of shadow tables if copying of rows from Sakai tables into shadow tables were desired prior to their deletion from the Sakai table
- require the addition of triggers to populate the specialized columns for every data manipulation language call for the Sakai tables
- (optionally) require the addition of triggers to the Sakai tables to copy rows to the corresponding shadow table prior to its deletion from the Sakai table in order to protect the users from accidental/premature deletions.
- (optionally) require the development, maintenance and periodic use of a shadow table cleanup routine.

On the surface it may sound as if adding these fields and shadow tables is a lot of work and perhaps even performance overhead. In actuality, the Audit and Hybrid Approach can (and has been) substantially automated\(^4\), so please do not trash this document just yet.

Unfortunately, the same thing cannot be said for the Early in Development Approach. While it does have its distinct advantages, its incorporation into Sakai would not be so easy.

Because it would most likely require extensive changes to existing application code, the Early in Development Approach would be best suited for Sakai modules that are early in their development.

Indiana University remains optimistic that both the Audit and Hybrid Approaches are viable. Given the fact that most database management systems read and write in relatively large blocks (which usually contain multiple rows) in combination with today’s powerful CPUs, fast disk drives and RAID technologies we are optimistic that for most Sakai environments, none of these approaches would negatively impact user performance noticeably. Our analysis of these approaches, while not nearly as exhaustive as we would like, has yet to show any substantial negative impact to user response.

The following chart displays Indiana University’s Estimated Resource Usage for each of the approaches for performing Archive/Purge. It assumes an annual archive/purge while retaining the most current three years of data. (100 represents the current level of resource usage.)

### IU’s Estimation of Resource Usage by Approach for Archive / Purge

Assuming the estimate is relatively accurate, then a little extra resource usage up front could pay substantial dividends in the future. The positive, long-term business impact of adopting such approaches (Early in Development, Audit, and Hybrid) would be:

- REDUCTION in necessary computing resources - Systems which appropriately manage data growth require a smaller storage, processing and memory footprint, ultimately leading to reduced ownership costs.
- IMPROVED system performance - Appropriately managed data growth, resulting in reasonably sized transactional systems, will improve system response time and more importantly, the intangible measure of end user satisfaction.
- REDUCTION in administrative overhead - For example, shadow tables alone provide a much simpler data recovery method than executing disruptive, full database restoration operations. Additionally, infrastructure staff would minimize the time spent managing system growth, storage allocations, etc. which can be cumbersome to any organization.

### Specialized Administrative Data:

**IU’s estimated Resource Usage for Archive / Purge**

![Graph showing resource usage over years for different approaches](Image)

100 represents the current level of resource usage
Conversely, and not surprisingly, the business impact of not adopting an approach similar to one of the aforementioned ones (Early in Development, Audit, Hybrid) would be:

- **INCREASE** in necessary computing resources - Unbridled data growth will ultimately require more storage, processing memory resources. Ultimately leading to the purchase of exceptionally large hardware to the tune of hundreds of thousands, if not millions of dollars for institutions of comparable size to IU. For example, within IU, utilizing just one fewer high-end processor would equate to a savings of roughly $20,000, which could be re-purposed to invest in other gems.

- **DECREASED** overall system performance - More data requires more execution cycles which means more time spent waiting for results by the customer. Slow system responses do not improve customer satisfaction.

- **INCREASE** in administrative overhead - More time will be spent by system and database administrators managing system resources particularly, around the art of performance tuning in both the application and database layers.

In summary, all three approaches are designed to ease the burden of providing the capabilities of archiving, purging, partitioning, integrating data in enterprise and departmental systems, report writing and optionally, recovering data lost due to user error. In order to accomplish this list, each approach would require the addition of some subset of the Specialized Administrative Columns (described earlier) which would not (necessarily) be directly used by the application but they would be used for the administrative and enterprise purposes mentioned above.

The Early in Development Approach would require considerable modification of the existing application to populate these columns. If the deactivation of rows were to be used instead of actual deletion of rows, then additional application logic would need to be written, however the use of deactivation would provide a relatively easy way to recover user data accidentally or prematurely deleted.

Neither the Auditing nor the Hybrid approach require the modification of the existing application code beyond the development of a routine to periodically purge data from the shadow tables. However, both would require the creation and maintenance of shadow tables and triggers to populate them. IU has developed code that would substantially automate these processes. That code is available to anyone in the Sakai Development Community.

Avoidance of Parsing

Parsing occurs when the datum in a column is accessed and the broken apart into smaller (discrete) pieces. This “breaking apart process”:

- adds computer processing time (slows response to the user)
- requires “extra” application code to be written and maintained
- requires report writers, enterprise infrastructure specialist and data integrators to (write) use equivalent code to the desired parsing.

If this parsing happens relatively frequently then it should be avoided by adding columns that would contain the discrete values being parsed from the columns.

If it is found that the functionality of the application frequently requires the concatenation of these discrete values, then adding a column for the concatenated values would be practical as well. However, storing only the “concatenated” value and parsing it to obtain the desired discrete values should be avoided.

Database design best practices call for storing one discrete value per field.

Several of the legacy tools in Sakai use a method of data storage that involves concatenation of several distinct data elements into a single string, separated by a delimiter character but does not store these elements in their own column. This, then, requires frequent parsing of this custom data structure to obtain the discrete element(s) needed by the application code.

For example, a single field in the SAKAI_EVENTS table contains strings like this:

/MessageCenter/site/11116bee-2d41-4ba4-801d-cc9f92140f6f/Message/2947064/01b25775-dec3-436b-8007-f4a2e9f3e7b6
Sakai: Building for the Future Now

Each of these items, separated by a ‘/’ character, are individual data elements that may be of interest for report generation. This particular string contains many items, including the name of the tool responsible for the event, the site id, the message forum id, and the message id.

Due to the custom data structure used in this example, an analyst tasked with writing reports based on this data would most likely have a more difficult time. An analyst would have to write (time-consuming) custom code that parses this specific structure. If discrete fields were used this job would most likely take only minutes to write in SQL.

Retooling of the existing application code to remove the need for parsing would be a major task since application code would need to be modified to take advantage of the columns resulting from separating the custom data structure into discrete columns. However, should the need/desire to re-write these Sakai legacy tools arise, then the opportunity to breakout these custom data structures into separate columns should be taken. Keeping a separate column to store the data structure would remain an option.

The following chart displays Indiana University’s Estimated Computing Resource Savings if the need for parsing non-discrete column values were eliminated. (100 represents the current level of resource usage.)

Resource Savings with Elimination Non-discrete Column Values

While difficult to estimate IU believes that additional benefits from avoiding parsing of non-discrete data would be:

- improved user response time
- easier, faster, and more accurate report writing and data integration.

The positive business impact of the eliminating or substantially reducing the frequent parsing of columns to obtain data critical to the functionality of Sakai would be:

- REDUCTION in costs associated with custom code – For any user, data analyst, report writer, etc. to make sense of embedded strings, they must first be educated to the meaning and location the data. Once educated, they must then customize their tool of choice to parse and consume the data. Repeated custom coding is not cost effective, let alone the cost of educating the user. In essence, Open Source business intelligence tools would be much more usefully leveraged if only one data element were captured per field.
- IMPROVED utilization of computing resources - The process of storing one data element per field is called ‘Normalization’. All commercial relational database systems are optimally designed to with ‘normalized’ data and as humans is how we understand data relationships. By appropriately modeling data elements within a Sakai module, improved performance with fewer computing resources in entirely feasible.
- BETTER data manageability - In the likely event that data needs to be manually manipulated due to human error, software issues or the like, it is a much simpler, predictable and much less time consuming task to update fields with individual data elements than embedded strings. This could potentially in significant saving to both development and database administration staff.

The Avoidance of Parsing: IU’s Estimated Resource Savings

• LIMITED analytical usefulness, if any at all - Due to the complexities noted earlier, embedded strings pose a very daunting challenge when considering the effort required arriving at useful analytical reporting. We all understand the inherent value in being able to consume and thoughtfully review data both at the operational and managerial level. Storing data within embedded strings hinders these types of activities and may cause institutions to overlook important trends in academic and business decision making.
• DECREASE in utilization of computing resources - Not to sound like a broken record, but the more data which much be parsed, the more intensive the computing resources required, particularly at the application server level.
• DIFFICULTIES in system integration activities - As SAKAI modules are generally developed independently of one another, there is very little guarantee that the data contained within embedded strings will be handled in
the same manner. Ultimately, this leads to increased development when performing both internal and external integrations. Utilizing a discrete data element per field would significantly simplify such processes.

In summary, as we continue to polish our Sakai gems, we should refrain (some might go so far as to say “be prohibited”) from using structures that require application logic to parse non-discrete data into discrete elements. The use of concatenated strings, serialized objects, and XML make the data in the database much less readable by humans and nearly impossible to parse by standard SQL report generation tools.

Human Recognizable Names

While the database and application cope very well with generic, abbreviated and/or obscure strings of characters, humans do not cope nearly so well with them.

For example, a column named HOST_ID in a table named HOST is more humanly recognizable than ID. COMPUTATION has more immediate meaning to a human than CMPTTN. The same would be true for OBSOLETE_1 as compared to OBS_1.

While the table and column names used in IU’s Sakai implementation are “pretty darned good”, there is a little room for improvement. This little improvement could significantly speed up the understanding and use of Sakai data by non-Sakai developers. A secondary benefit would be that both new and less experienced developers would come up to speed a little faster.

At Indiana University we have seen the need for report writers and data integrators (who may not be as technically savvy as developers) to interact with the database to produce metric reports and other forms of analysis. The use of consistent naming standards throughout the database eases the understanding of report writers and data integrators. This in turn allows the faster, more accurate production of reports and analysis. A secondary benefit would be that both new and less experienced developers would come up to speed a little faster.

In the past, most databases consisted mainly of simple tables and indexes. These include regular data tables, views, materialized views, partitioned tables, shadow tables, unique indexes, non-unique indexes, partitioned indexes, bit map indexes, etc. The usual method of making these objects humanly recognizable is to adopt standard suffixes like:

- Tables end in _T,
- Partitioned tables end with _P<nn>
- Views end with V,
- Materialized Views end with _MV
- Shadow Tables end with _X

(<nn> represent a two digit number>)

By using such suffixes, individuals (Sakai developers and non-Sakai developers) can easily identify the type of the database object which in turn drives an easier identification of how to best access the data in that database object.

It is noted that in today’s Sakai application not all of the different types of tables and indexes mentioned above are in use. However, the future may see them pressed into service. By adopting such standard suffixes, we will be building now for the future. This will help eliminate or reduce future rework.

When we look into the underlying database of the Indiana University Sakai application we find that: 139 out of 368 of the tables end with _T.

We also find that out of the 741 indexes used by Indiana University Sakai and application, all of them ended with a suffix indicating the type of index. Unfortunately, we have not always been consistent with the actual suffix used. Some ended with INDEX, some with IDX and others with I<nn>.

So, if you are building a new Sakai module or are in the early development stages of a module, we highly recommend that you identify standard suffixes and use them consistently throughout your module.

If you find yourself in the position of being able to consider retrofitting standard suffixes, renaming tables and indexes inside the database is a relatively small task. The larger task would be the renaming of the tables in the application code. Based on discussions with multiple Sakai developers, here is how one might go about renaming the tables in the application code.

In the case of modules that are built on top of hibernate (or similar facility), it should be fairly straightforward to rename tables. The names need to be changed inside of the hibernate mapping file (HBM file). This will automatically cause hibernate to use new names for any new installations. The DDL for table and field renames needs to be added to the release conversion script so that institutions upgrading from previous versions have a specification to rename their existing tables.

The worst case scenario is that the table names referenced in the application will need to be edited. This would be a daunting task that would burn some “precious time” and resources.
Sakai: Building for the Future Now

Another, and perhaps more pressing need for using human recognizable names is in the naming of columns. One of the most basic axioms of database design is that columns that measure/indicate the same thing should be named the same. Conversely, columns that do NOT measure/indicate the same thing should be named differently.

In IU’s Sakai application today, we have a column named ID in 115 different tables. Not all of these columns measure/indicate the same thing. While this will not cause any negative impact within the database or the application code, it could easily confuse an individual accessing the database from outside of the application.

IU has received several complaints about how difficult it is to query the database when one cannot easily tell which ID columns are the same and which ones are different. It is true that one can query the meta data to make this determination but that takes time, database experience, and the appropriate privileges and security levels to query the meta data. Not all Institutions may want to grant such privileges and/or security levels.

Another issue with the generic use of ID as a column name is some of the columns named ID are primary keys. Sometimes the primary keys have foreign key references in child tables. Database design best practices call for primary keys and their associated foreign keys to be named the same (or as similarly as possible). This cannot be done if the child table already contains a column using the generic name. Individuals accessing the underlying database cannot visually see the keys and cannot easily see the join that may be necessary. Worse yet, they may assume that the ID in the child table is the foreign key to the parent table that has ID as its primary key and do an incorrect join.

One way to avoid these issues is to append the table name in front of the generic ID. For example, in a table named MYSTUFF the column name would be MYSTUFF_ID not just ID. Then the foreign key in the child table named MYSTUFF_CHILD would be humanly recognizable as MYSTUFF_ID and still allow one to use the column name of MYSTUFF_CHILD_ID without confusion caused by just using ID.

In other words, please try to avoid using a column named ID instead name it <TABLE_NAME>_ID. When it is a primary key, and then insure it is referenced in the child table as <TABLE_NAME>_ID as well.

Renaming the columns that are named ID inside the database is a relatively small task. Again the challenge lies in renaming the columns inside the application code.

Just as it is with renaming tables, in the case of modules that are built on top of hibernate (or similar facility), it should be fairly straightforward to rename fields. The names need to be changed inside of the hibernate mapping file (HBM file). This will automatically cause hibernate to use new names for any new installations. Then DDL for table and field renames needs to be added to the release conversion script so that institutions upgrading from previous versions have a specification to rename their existing tables.

To illustrate the value of using human recognizable names, please refer to figure 1:

Figure 1 contains four (out of five) tables that use the generic name ID – all with the same data type and precision. However, in spite of being named the same and having the same data type, each measures/indicates something different.

While a current Sakai developer may not have any issues with figuring out how these tables relate to one another, a less experienced developer, report writer or data integrator would probably find the relation amongst these tables to be somewhat challenging. Indeed, some might conclude the obvious and attempt to join the tables based upon the ID columns. After all, an axiom of database design is that columns named the same are the same. In this instance, this would not produce the desired results. Even if it were recognized that the ID columns are not the same, the individual accessing these tables would need to query the database’s meta data to determine the appropriate relationship – if s/he has the correct privileges. This adds time, effort and complexity to the process. Much of which can be substantially reduced (or even eliminated) if human recognizable
Sakai: Building for the Future Now

names were used. See Figure 2: Generic ID Columns Renamed to be more humanly recognizable.

The difference between Figure 1 and figure 2 is that generic ID columns were renamed to indicate their table and these new names (or ones very similar) were used to rename the foreign keys in associated child tables.

By using more humanly recognizable names -- report writers, data integrators and less experienced developers can more easily see the relationship that ONC.NODE_T is the parent table to SITEHIERARCHYASSOC, AGENTGROUP and NODEPARENTCHILDMAP. This also eliminates any confusion that may have resulted from using ID within each of the tables.

The following chart illustrates one of the major benefits of using Human Recognizable names. It displays Indiana University’s Estimated Coding Time Needed for Report Writers and Data Integrators before and after renaming. (100 represent the current level of coding time.)

IU’s Estimation of Report Writer/Data Integrator Coding Time Required Before and After use of Human Recognizable Naming

While your estimate may vary, we believe that a little up front effort in using human recognizable names to polish your Sakai Gems would pay off substantially in the future. This would be especially true when it comes time to placing your gem in its setting – your enterprise infrastructure.

The positive business impact of eliminating the use of the generic ID column and using more humanly recognizable names within the underlying Sakai database would be:

- **REDUCED development effort** – By uniquely identifying database tables and keys in a standardized fashion, the use and relationships of table data becomes much more obvious to the naked eye. Such a change will facilitate reduced development time and allow for easier integrations, by guaranteeing field uniqueness within the database. Reduction in development time ultimately leads to reduced overall cost OR increase functionality.

- **IMPROVED long-term database viability** - If the Sakai communities were to adopt a set of database naming conventions, the long-term viability of the database will be greatly improved. It is ostensible that the Sakai database will only grow in terms of tables and number of modules supported. If this growth is not managed with an organized and standardized approach, there is small hope of being able to sort through the database structure in the event of troubleshooting, performance or data integrity issues. Additionally, integrations will become increasingly difficult, particularly between modules which do not represent field names in an identical manner.

The business impact of NOT eliminating the use of the generic ID column and NOT using more humanly recognizable names within the underlying Sakai database would be:

- **CONTINUAL DECREASE in developer productivity** – If the number of Sakai database structures continue to grow without adoption of a formal standard, it is highly likely that much more time will be spent on system integrations as well as data manipulation activities. Development staff will need to wade through and compensate for challenges posed by a lack of non-distinct column names and proper parent/child relationships.

- **INCREASED support complexity** - As we are all aware as software systems mature grow the feature set, the overall all complexity of the solution also tends to significantly increase. If the complexity of the solution makes supporting the solution a difficult chore, then typically ‘simpler’ solutions are then sought out as alternatives. (Simpler solutions typically more cost effective.) If Sakai takes small efforts now to help minimize complexity, this issue can be proactively addressed.

The Use of Human Recognizable Names:

**IU’s Estimated Coding Time Needed for Report Writers and Data Integrators Before and After Renaming**

In summary, if the Sakai Application were stand alone and there would never be any need to access its data from outside of the application, the use of humanly recognizable names (while still desirable) would not be as large of an issue. However, at Indiana University, we have found the Sakai data has been needed by other systems within our enterprise. We have noted that individuals like DBAs, SysAdmins, Data Integrators and Sakai Support Personnel have been called upon to access, report, extract and/or integrate Sakai data. The use of humanly recognizable names would speed their understanding of the data and therefore allow them to process Sakai data faster and more accurately. The same benefits would be realized by less experienced Sakai developers and developers new to Sakai.
Sakai: Building for the Future Now

We predict that this will be true for many, if not most, of the Sakai community. Therefore, we urge you to “shape and polish your Sakai gems” by planning and using humanly recognizable names that are consist with database design best practices.

So for a little effort now, the shaping and polishing of your Sakai gem (module) now by using human recognizable names will save time, resources and money in the future when it comes time to fit it into the Sakai Ring (your enterprise infrastructure).

Closing

In closing, we at Indiana University, believe that if the Sakai Development Teams at our member institutions will give due consideration to the needs of the ring setting (administrative and enterprise infrastructure) while cutting, shaping and polishing their Sakai gem(s), the results will be admired for the beauty of its gems as well as their arrangement within the ring.

Your comments and feedback are encouraged. Please forward them to: damckee@indiana.edu

About the Author:

Dan holds a Bachelors in Science and Masters in Educational Psychology from Ball State University and a doctorate in Statistics and Policy Research from the University of Cincinnati.

He has been a science teacher, research analyst, statistics instructor, functional analyst and database designer.

Prior to joining Indiana University’s Sakai Project in 2004, Dan spent 25 years designing enterprise databases for the pharmaceutical and telephony industries.

End Notes:

1 For example: “My assistant misunderstood me and deleted the wrong sites. Can you get them back for me?”
2 Currently IU has not found a way to capture the user making the modification to the row.
3 This would be case where the application is using the database’s cascading ability to delete child rows.
4 IU is continuing to develop such automation code and it is available to the Sakai community.
5 IU has direct experience that validates this assertion.
6 Specialized tables used for data recovery and/or auditing.
7 Security restriction(s) may apply.