

Out in Left Field

Volume IV, Issue VI

THE KEYS TO THE KINGDOM

by Daniel Cruikshank

I would like to take some time to discuss the use of keys when designing new databases or re-engineering existing databases. I believe the database is the foundation for any application. A good relational database design requires the use of keys, also referred to as constraints. The implementation of database keys is an essential piece to constructing safe and secure databases. In this article I will discuss the characteristics and uses of the following key types:

- Unique Keys
- Primary Keys
- Foreign Keys

The Unique Key

Although not required, each and every base table (physical file object on IBM i) should have at least one unique key defined. A unique key identifies a single row within a table to the application. Note that I said application, not database. I'll discuss this later in the article. The main purpose of the unique key is to allow the application to retrieve, update or delete a single row within a table.

The following are characteristics of a unique key:

- One or more per base table
- Can be null
- One or more columns
- Any data type
- Is typically a known value (i.e. it is not hidden)
- May be used to establish referential constraints

A unique key can be created using the CREATE UNIQUE INDEX DDL statement or by using the UNIQUE constraint clause of the CREATE or ALTER TABLE DDL statements. Although you can create multiple unique

constraints per table, I prefer using a constraint to identify the *row unique key*. I can always create additional unique keys using the CREATE UNIQUE INDEX DDL statement.

A sequence object can be used to auto generate the next row unique key value. Although sequence objects contain numeric values, you can wrap a CAST around the output of the NEXT VALUE statement to create alpha-numeric unique values. You can read more about sequence objects at the IBM System i and i5/OS Information Center web site located at url <http://publib.boulder.ibm.com/iserics/>.

You can also find an article on sequence and identify columns at <http://www.centerfieldtechnology.com/publications/archive/April%202006.pdf>

In some cases, especially within validation and/or lookup tables, the row unique key attributes are determined by what is considered conventional and/or customary usage (also referred to as a natural key). For example, a country code lookup table may use the ISO standard two character abbreviation for a country. In this case, the country code would be the row unique key. The country name would be a secondary or alternate unique key (you would not want two country codes for the same country).

For discussion purposes, let's assume we are creating a database to store data about a household. Within a household there are many different subjects that need to be defined to the database. For example the house itself, the house's occupants, the occupants' possessions, the different locations associated with the house, the items contained within each location, etc. We start by creating a logical data model (LDM).



(Continued on page 3)

Inside this issue:	
Kings to the Kingdom	1
SQL Stored Procedures	2
Choosing between hardware and software encryption	12



SQL Stored Procedures



One of my colleagues worked with a customer this month with a .NET application invoking about 4000 SQL stored procedures on the System i. I helped a customer who has a Java Web application which alters the database exclusively through SQL stored procedures. I talked with an IBM DB2 guru who recently assisted a couple of vendors port their applications from other database platforms that use massive amounts of SQL stored procedures (many thousands).

With this seeming popularity of SQL stored procedures, one question that continuously comes up once functional obstacles are overcome was 'how can we make them perform better'.

NOTE: For purposes of this article, **SPL-SP** stands for **SQL Procedural Language-Stored Procedure**.

Cover the basics

As with any application running a significant SQL workload, the best starting place for performance tuning is a good indexing strategy (See Appendix). Having a sufficient number of 'perfect' indexes helps the query optimizer provide an abundance of real time statistics and alternative query implementation methods.

So let me repeat, a good indexing strategy is an ABSOLUTE MUST to get your application performing up to par. If you don't complete this optimization step and your application database is of any appreciable size, no amount of application and SQL tuning will matter.

Another common pitfall that we used to see in the past was that the machine was simply undersized to run SQL workloads efficiently. Thankfully, this has not been a problem for a number of years now. Hardware power has skyrocketed, its costs have gone down and most importantly, hardware sizing tools and business partners have become more aware of the SQL workload demands.

Instead, what we occasionally see is poor allocation of existing memory resources, incorrect application of the dynamic performance adjuster, poor journaling configuration and the like. All of these obstacles are easily addressable through simple reconfigurations. Knowing if you need expert help in this area will take you a long way toward resolving the issue.

(Continued on page 9)

Below is a screenshot of the household LDM defined using the IBM Rational Data Architect product:

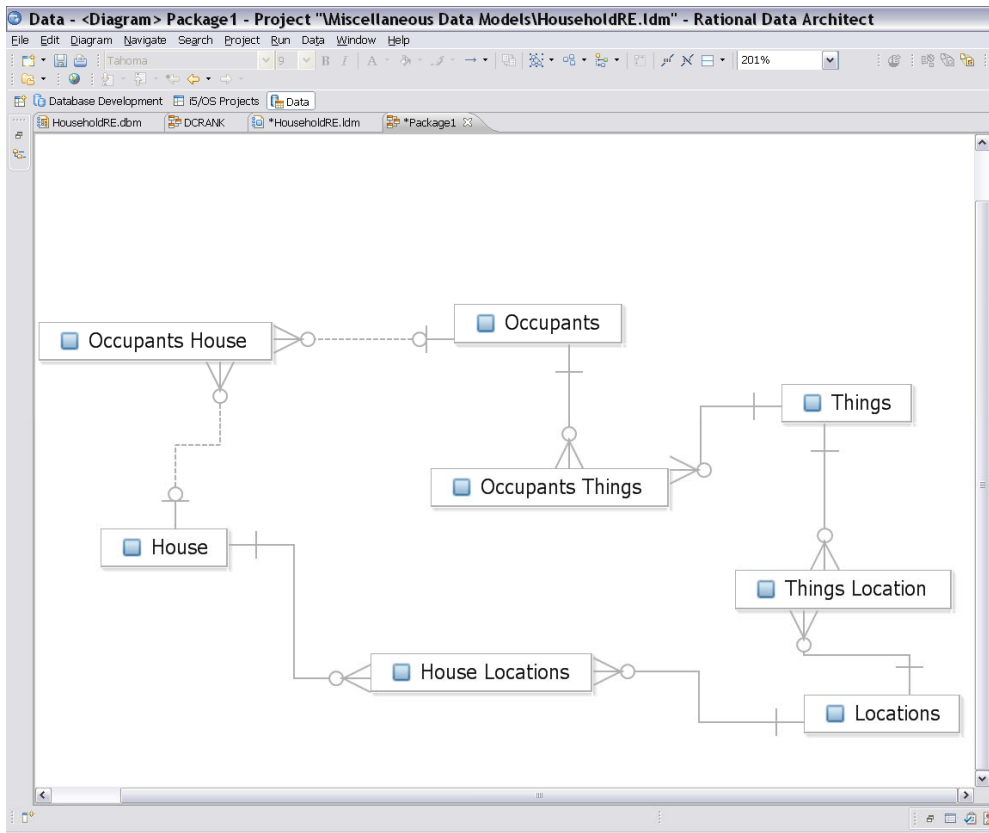


Figure 1 Household Logical Data Model using RDA

Next we define our base tables. Ideally, the same tool we used for creating our LDM would be used for this process. For the purposes of this article I have chosen the System i Navigator as the tool for defining and creating the DB2 for i database objects. One main benefit of using this tool is that it is IBM 6.1 release ready. In essence, you can immediately begin using new database 6.1 features and functions when defining your database objects. The included DDL examples in this document were generated from System i Navigator.

Below is an example of the DDL that would be used to create the HOUSE table:

```
CREATE TABLE HOUSE (  
  HOUSE_NUMBER INTEGER DEFAULT NULL ,  
  HOUSE_COLOR VARCHAR(30) DEFAULT NULL ,  
  HOUSE_STREET_ADDRESS VARCHAR(30) NOT NULL ,  
  HOUSE_POSTAL_CODE CHAR(10) DEFAULT NULL ,  
  CONSTRAINT HOUSE_UNIQUE_KEY UNIQUE KEY(HOUSE_NUMBER) );
```

A unique key constraint is defined for HOUSE_NUMBER which makes it the row unique key. The HOUSE_NUMBER column is visible to the application and can be used for updates and/or deletes and occasionally for inquiries when the house number value is known.

(Continued on page 4)



Dan Cruikshank, Senior Consultant, IBM Systems and Technology Lab Services Group
Dan has spent the last 19 years with IBM and has over 35 years I/T experience. His areas of expertise include application and database performance troubleshooting and optimization. He has developed many IBM internal tools for extracting and analyzing system and database performance data. He has written extensively on application performance tuning and database reengineering (DDS to DDL). He is currently one of the IBM Rochester certified instructors for the IBM i Performance Tuning Workshop.

Below is an example of using the house unique key in an UPDATE statement:

```
UPDATE HOUSE SET HOUSE_COLOR = 'Green' WHERE HOUSE_NUMBER = 1
```

In many cases the row unique key may be known externally by an end user. For example, many people know their employee number which may be the row unique key for the employee table. In this case of our household database, the HOUSE_NUMBER value may or may not be known which means that an alternate unique key must be created to satisfy simple searches.

The following is an example of adding a secondary or alternate unique key:

```
CREATE UNIQUE INDEX HOUSE_ADDRESS ON HOUSE
(HOUSE_POSTAL_CODE, HOUSE_STREET_ADDRESS);
```

This new key combines the postal code with the house street address. It is mainly used for inquiries. Entering the postal code and full street address returns a single row. Entering a postal code and a partial street address may return one or more rows. In either case, the house row unique key is always returned to the application for subsequent access.

The above examples of unique keys may or may not be realistic, I am mainly using them to make a point and that is that a unique key can be known, either inadvertently (by displaying or publishing the HOUSE_NUMBER) or by design (for instance the HOUSE_POSTAL_CODE and HOUSE_STREET_ADDRESS). In some cases, the unique key may contain sensitive data and requires encryption (for example an email address used for occupant lookups). The last thing that should be done with this type of unique key is to propagate it throughout the database as a foreign key. This is a role better suited for a primary key.

The Primary Key

Knowing the unique key of a house and knowing what is contained within the house are two very different things. Using the house address, you can get driving directions directly to the door of the house, but this is not going to get you inside. Nor is digging through the owner's trash and finding the published house number going to unlock the door. You will need either the owner's permission or the house key to enter the house. In this database I refer to the house key as the *primary key*. The primary key is used to uniquely identify each row within a table throughout the database. Based on our LDM we learned that the house entity is associated with several other entities (OCCUPANTS, LOCATIONS, etc). The primary key defined for each entity is used to create relationships between other entities.

The following are characteristics of a primary key:

- One per base table
- Cannot be null
- Is defined as a table constraint
- Best practice is a single numeric column
- Meaningless
- Can be auto generated via Identity Column support

The following table compares the recommended characteristics of a primary key to a unique key:

Characteristic	Primary Key	Unique Key
More than one per base table	No	Yes
Can be null	No	Yes
Must be a constraint	Yes	No
Can be known to the outside world	No	Yes
Contains meaningful data	No	Maybe
Needs to encrypted	No	Yes, if meaningful data is sensitive
Can be used in a referential constraint	Yes	Yes, however this should be avoided if data is sensitive
Should be auto generated	Yes	Not always

(Continued on page 5)

(Keys to the Kingdom - Continued from page 4)

A primary key can be created using the PRIMARY clause of the CREATE TABLE DDL statement, or by using the PRIMARY KEY clause of the ALTER TABLE DDL statement. Ideally, the attributes of the column designated as the primary key should be numeric. This allows the primary key column to be an identity column. Unlike a sequence object, the identity column support is part of the table object. To learn more about identity column support check out the DB2 for i Articles and White Paper support page at url <http://ibm.com/systems/i/software/db2/awp.html>.

The following is an example of defining an identity column as a primary key during the creation of the HOUSE table:

```
CREATE TABLE HOUSE (
  HOUSE_KEY FOR COLUMN HOUSEKEY BIGINT
  GENERATED BY DEFAULT AS IDENTITY
  IMPLICITLY HIDDEN - (V6),
  HOUSE_NUMBER INTEGER DEFAULT NULL ,
  HOUSE_COLOR VARCHAR(30) DEFAULT NULL ,
  HOUSE_STREET_ADDRESS VARCHAR(30) NOT NULL ,
  HOUSE_POSTAL_CODE CHAR(10) DEFAULT NULL ,
  CONSTRAINT HOUSE_PRIMARY_KEY PRIMARY KEY(HOUSE_KEY),
  CONSTRAINT HOUSE_UNIQUE_KEY UNIQUE( HOUSE_NUMBER )) ;
```

The following is an example of adding an identity column and primary key constraint to an existing table:

```
ALTER TABLE HOUSE
  ADD COLUMN HOUSE_KEY BIGINT
  GENERATED BY DEFAULT AS IDENTITY
  IMPLICITLY HIDDEN --(V6)

ALTER TABLE HOUSE
  ADD CONSTRAINT HOUSE_PRIMARY_KEY PRIMARY KEY (HOUSE_KEY) ;
```

When defining an identity column I tend to use a BIGINT which only uses 8 bytes of storage but allows a maximum value of 9,223,372,036,854,775,807. Probably overkill for the HOUSE table but something I will never have to worry about changing in my lifetime. Generating the values by default allows me to copy the HOUSE table and retain the existing primary key values. This allows a copy of the HOUSE table to be used for testing purposes once relationships have been established to other tables within the database.

Because the primary key is used strictly within the database I take advantage of the new 6.1 IMPLICITLY HIDDEN clause to hide the primary key from the application. Using this clause as part of a column definition prevents that column from being returned unless it is explicitly named in the result set of an SQL statement. This prevents data from being displayed by queries which request all columns from a table (i.e. using SELECT *). There is no need for the application to know the value of the primary key or the complexity of the database. This may also eliminate the need to encrypt the primary key value. It is important to note that some non-SQL interfaces do not honor the implicitly hidden attribute. You can read more about what's new in 6.1 at the IBM System i and i5/OS Information Center web site located at url <http://publib.boulder.ibm.com/series/>.

As stated previously, the household database is made up of several subject tables: the house table, the occupants table, the locations table, etc. Once each separate table is created, with the appropriate keys, relationships are established by defining Referential Integrity (RI) constraints between the tables.

The Foreign Key

The base table and the primary or unique key being referenced in the RI constraint are referred to as the parent table and parent key respectively. The base table and the key referencing the parent table and parent key in the RI constraint are referred to as the dependent table and *foreign key* respectively. In essence, a foreign key is a field in a dependent table which contains a copy of the data contained in the primary or unique key field defined within the parent table. The foreign key is typically not unique, depending on the relationship between the two tables making up the referential constraint.

For example, a one-to-one relationship may exist between the HOUSE table and the HOUSE_PROPERTY_VALUES table. The data contained in the HOUSE_VALUE table is sensitive and highly volatile whereas data like the HOUSE_STREET_ADDRESS may never change. Although the data could be contained in a single table, separate tables provide an additional layer of security and reduce index maintenance and journal overhead as the house market value is constantly changing. In this case, the foreign key could also serve as a unique key to prevent duplication within the HOUSE_VALUE table.

(Continued on page 6)

(The Keys to the Kingdom — Continued from page 5)

The following are characteristics of a foreign key:

- One or more per base table
- Can be null
- One or more columns corresponding to like columns in the parent table

A foreign key (or referential constraint) is created using the FOREIGN KEY clause of the ALTER TABLE SQL statement. Below is an example of creating a foreign key constraint between the HOUSE and the HOUSE_OCCUPANT_XREF tables:

```
ALTER TABLE HOUSE_OCCUPANT_XREF
ADD CONSTRAINT HOUSE_OCC_XREF_FOREIGN_KEY
FOREIGN KEY( HOUSE_KEY )
REFERENCES HOUSE ( HOUSE_KEY )
ON DELETE CASCADE
ON UPDATE NO ACTION ;
```

I always name the foreign key the same as the corresponding column (or columns) in the parent table. This allows me to take advantage of the shorthand USING clause when writing an SQL join statement. If you do not define a constraint name (**CONSTRAINT** clause) then DB2 for i will generate a name for you. I prefer creating my own names which aids in subsequent database analysis.

A discussion of the database actions (**ON DELETE, ON UPDATE**) associated with an RI constraint is outside the scope of this article. The role RI plays during optimization is discussed later in this article. Again, refer to the DB2 for i and IBM System i and i5/OS Information Center web sites for more detailed information.

The following System i Navigator screenshot displays the existing constraints for the Household database:

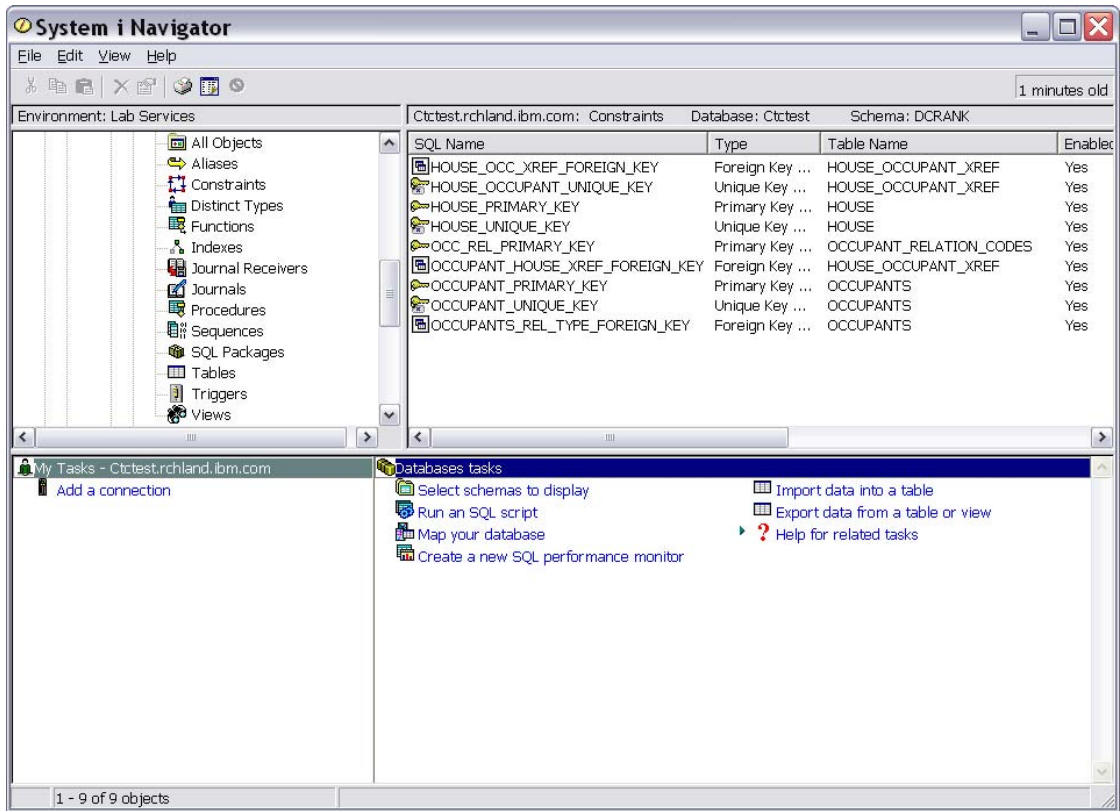


Figure 2 System i Navigator Constraints View

Putting it back together

I use a concept of associative tables (aka linkage or cross-reference tables) to create relationships between two tables in a many-to-many relationship. For example, a house can contain many occupants and an occupant could be a resident of multiple houses

(The Keys to the Kingdom — Continued from page 6)

(main residence, summer cabin, winter condo, etc) although physically not at the same time. The associative table can also reduce the overhead of index maintenance and journal logging as the occupants things could constantly be moving from house to house, one location to another or changing ownership between occupants.

The associative table typically contains nothing but foreign keys, however at least one column of a base table must not be implicitly hidden , which means a possible foreign key value exposure.. One solution to this potential problem in 6.1 is to add a non-hidden field to the associative table, for example HOUSE_RESIDENCE_TYPE, and make all foreign keys implicitly hidden. Unfortunately this is not available prior to 6.1. To get around this I mask the complexity of the database through the use of SQL views. This allows me to pick and choose the non-hidden columns which are required by the application.

Below is an example of creating an SQL join view which only displays the non-hidden columns from the data tables and excludes the foreign key columns from an associative table:

```
CREATE VIEW HOUSING_OCCUPANTS
AS
SELECT T1. HOUSE_NUMBER , ..., T2.OCCUPANT_NUMBER, ...,
       T3.OCCUPANT_GENDER
       FROM HOUSE T1
       JOIN HOUSE_OCCUPANTS_XREF X1 USING (HOUSE_KEY)
       JOIN OCCUPANTS T2 USING (OCCUPANT_KEY)
       JOIN OCCUPANT_RELATION_CODES T3 USING (OCCUPANT_RELATION) ;
```

In the above example I use correlation names to project the non-hidden columns from tables T1, T2 and T3. This allows me to exclude the columns from the associative table HOUSE_OCCUPANTS_XREF (X1). This table contains non-hidden foreign keys corresponding to the HOUSE and OCCUPANTS tables. Since X1 is not in the projected result set, the values contained within the foreign keys are not visible to the application.

For example, executing the following query against the HOUSING_OCCUPANTS view:

```
SELECT * FROM HOUSING_OCCUPANTS
WHERE HOUSE_POSTAL_CODE = 80403 AND HOUSE_STREET_ADDRESS = 'Hawaii';
```

produces the following result:

House Number	Street Address	Postal Code	House Color	Occupant Number	Occupant Name	Occupant Relation	Occupant Gender
259	Hawaii	80403	null	134107	Jules	Son in law	M
259	Hawaii	80403	null	922908	John	Grandson	M
259	Hawaii	80403	null	375213	Rich	Dad	M
259	Hawaii	80403	null	726505	Chance	Uncle	M
259	Hawaii	80403	null	529197	Rebecca	Daughter in law	F
259	Hawaii	80403	null	152305	Drusilla	Daughter	F

The unique keys for the HOUSE and OCCUPANTS (HOUSE_NUMBER and OCCUPANT_NUMBER respectively) are visible to the application and can be used for subsequent single row retrieval. The primary and foreign keys are hidden and only known to the database.

(The Keys to the Kingdom - Continued from page 7)

Optimization

Many of you are aware that the DB2 for i Optimizer relies on indexes for statistics and implementation. The optimizer is also constraint aware. Primary, unique and foreign key constraints are implemented as radix indexes under the covers on the System i. This means the optimizer can both extract valuable information from constraints (cardinality, average duplicates for join costing, etc) and subsequently use the constraint to implement the query.

Below is the Visual Explain of an UPDATE statement used to change the house color of a single row in the HOUSE table:

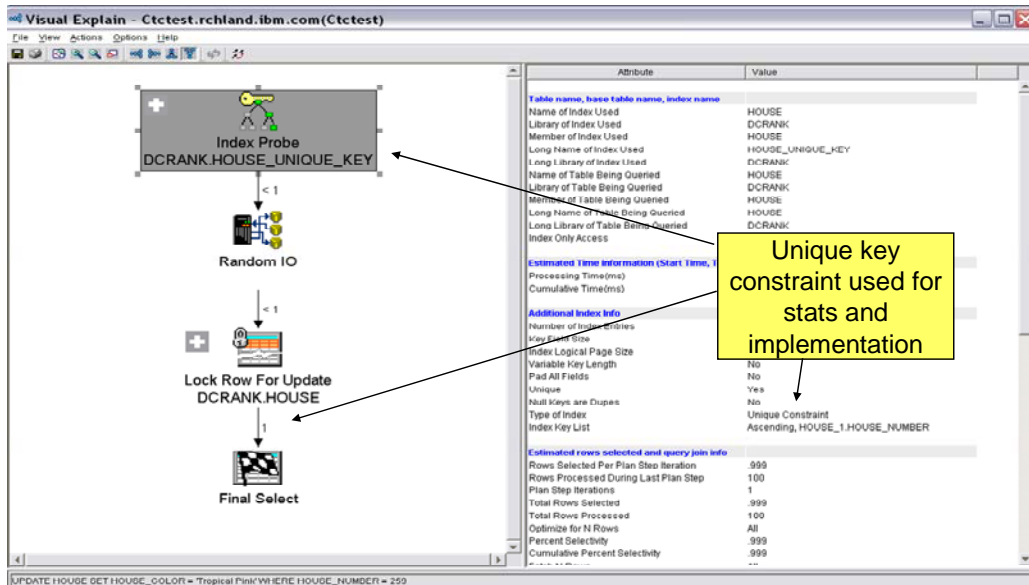


Figure 3 Visual Explain of Unique Key Constraint

Referential constraints provide dual capabilities. One, they are an essential building block for data centric application development and two, they provide an automatic join path when implementing nested loop joins using indexes. This is critical for high volume transaction processing applications.

Below is the Visual Explain of the house address search query shown previously:

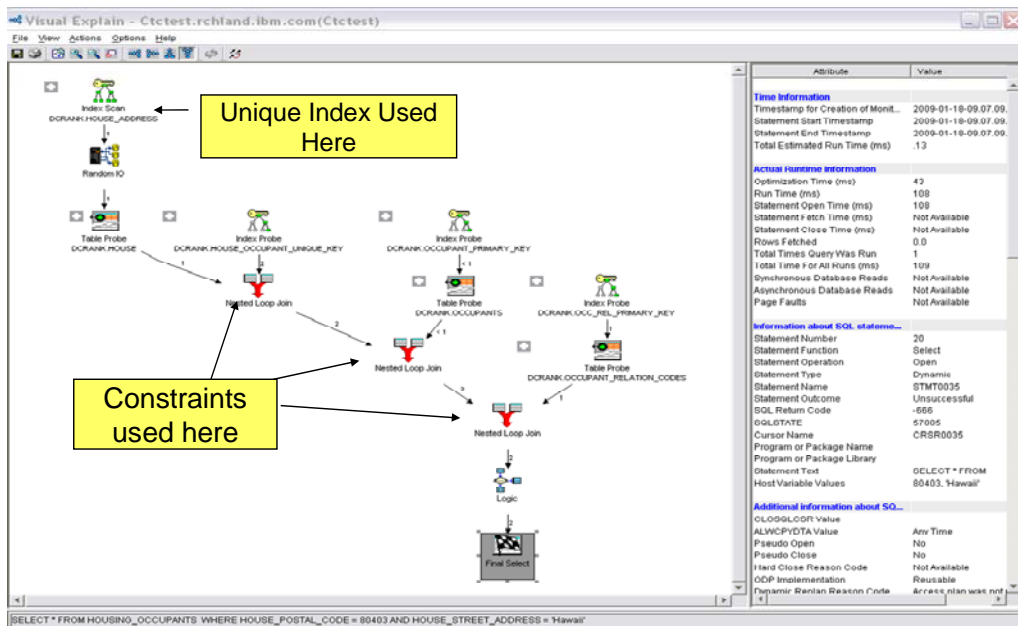


Figure 4 Visual Explain of Join using Constraints

(SQL Stored Procedures—Continued from page 2)

Avoiding single statement SQL stored procedures

Some of the benefits offered by stored procedures are: a reduction in communication line traffic, localizing application logic on the server, balancing computational workloads between different solution components, making remote program calls in an easy and standardized manner, server side enforcement of security policies and leveraging static versus dynamic SQL.

These benefits may cause a particular shop to standardize all database access through stored procedures. This is often the case with web and multi-tiered application solutions, and at times it is inevitable. However, it can also lead to an extreme where database developers code an abundance of single statement stored procedures. I cringe when I see these, because the overhead of invoking a stored procedure for a single SQL statement is significant and accumulates very quickly if the stored procedure turns out to be popular. A feasible alternative would be to create an SQL view and have the application code query the view directly. Any security rules could be addressed using native IBM i security or by using equivalent SQL privileges.

Recreating SPs on release boundaries

IBM has a long standing tradition of running your code from 20+ years ago on new versions of the operating system without any action required on your end. While this is an amazing feature of the IBM i operating system (kudos to IBM), I must tell you that is not always the case with SPL-SPs. Some of the recent SPL enhancements are only available through enhanced compiler support, which is release specific and requires you to recreate the SP. Thus, one of the standing recommendations I make to our customers who have significant SPL-SP workload is to recompile (recreate) all of the SPL-SPs when they move to a new version of the operating system. I realize that a simple save/restore process *works*, but you're losing out on potentially significant compiler optimizations by not recreating your SPL-SPs. The benefits of going through this simple process pay off very quickly, and quite frankly, I find them to be a no-brainer.

With IBM 6.1 I would also encourage this step after the database group #5 is loaded. Needless to say, if you don't already have a simple SQL script to recreate all of your SPL-SPs, now is the time to implement one. Perhaps you can instrument your script to use the new ALTER PROCEDURE command, available with V5R4. In fact, you could probably write a simple program that retrieves the list of SPs from the SYSPROCS SQL catalogue and loops through all of them, running something like: `ALTER PROCEDURE storedProcedureName NOT FENCED;` against them. This would assure any existing privileges and authorities are preserved.

(Continued on page 10)

(Keys to the Kingdom— Continued from page 8)

Summary

My main goal of this article was to make a clear distinction between primary and unique keys. Many relational database design books may use primary and unique keys synonymously. In the past I also never gave this much thought. My thinking on this has changed dramatically in light of the importance data now plays on everyday life. Sensitive data must be protected both externally and internally.

Remember these simple distinctions between keys:

1. A primary key is used to establish relationships and must be hidden from prying eyes
2. A unique key is used to locate data and therefore must be exposed.
3. A meaningless primary key value makes the best foreign key.



Remember, as a database architect, you are responsible for the “keys to the kingdom”.

(SQL Stored Procedures—Continued from page 9)

Implement best coding SQL stored procedure practices

In a couple of recent engagements we've had, I have realized that more information is needed to fine tune actual stored procedure code written in SPL. As I set on my search to accomplish this feat, I recalled a whitepaper Jarek Miszczyk and Gene Cobb wrote pertaining to new V5R4 Expression Evaluator support in SPL. This excellent paper is titled "Improving SQL procedure performance" and you can obtain it at this link: <http://www.ibm.com/servers/enable/site/education/wp/18A06/18A06.pdf>

It has since been updated with porting work done by Kent Milligan and includes some 6.1 enhancements. There is a lot of valuable, non-trivial advice in this paper. I believe our readers can benefit from it, so I am going to try and extract some of this advice which you can relatively easily implement in your SPL-SPs. I took a lot of liberties in interpreting and summarizing the advice, so original authors will just have to forgive me if I made mistakes:

Advice	Recommendation	Reason	Value
CLOB variables in SET assignment statement	Avoid	Prevents usage of Expression Evaluator	Medium
UDF in SET assignment statement or IF-CASE-WHEN comparison statement	Avoid	Prevents usage of Expression Evaluator	Medium
SQL query in a SET assignment statement	Avoid	Prevents usage of Expression Evaluator	Medium
CQE implementation	Avoid	Can't benefit from Expression Evaluator	High
Data Conversion	Avoid	Data conversions incur at least a 15% performance hit	Medium
Service program objects (6.1 only)	Use	If you add the PROGRAM TYPE SUB attribute to your CREATE PROCEDURE statement, you'll be creating a service program instead of a program object. This will give you a slight performance boost.	Low
Calls to other SPs	Avoid	SPL makes unbound program calls which creates additional performance overhead. Avoid them if you can, otherwise try to minimize them.	Medium
Scope condition handlers	Use	You can scope a specific condition handler just to the condition/statement it is handling. This is done by wrapping the condition/statement with the BEGIN and END clauses.	Low
Combine sequences of complex SET statements that leverage Expression Evaluator into one statement.	Use	Analyze SET statements using Expression Evaluator (database monitor) and try to minimize them by combining multiple SETs into a single SET statement. You can separate multiple assignments with comma (,) operator.	Medium
Temporary variables	Avoid	Avoid using temporary variables. Try to combine multiple statements and/or calculations into a single statement where possible.	Medium
Character and Date data types	Avoid	Developers sometimes use temporary variables for 'Yes' and 'No' Boolean type flags. Use Integer data type instead, as database engine does no CCSID and DATFMT validation for Integer data type.	Medium
Decimals variables with zero scale	Avoid	Use Integer variables instead. In fact, use Integer or Bigint variables whenever possible.	High
Tag local Character variables with 65535 CCSID	Use	For Character variables used only locally in the SP (i.e. you're not going use them to update the database). Tagging character variables ensures database doesn't invoke its translation code (65535 = *HEX = binary, which means it's not translatable).	High

(Continued on page 11)

(SQL Stored Procedures—Continued from page 10)

In addition to the advice extracted above, I also encourage adding the `SET OPTION DBGVIEW=*SOURCE` attribute to your CREATE PROCEDURE statement. This will ensure that you have debug information embedded with the program. You can then look through the debug listing and see which assignment and comparison statements are implemented in C versus one of the other 3 methods (QSQVALUE call, Expression Evaluator, SELECT against QSQPABL). Of course, you can peruse the compiler output to find this out.

Finally, use the database monitor queries outlined in the paper to gauge which assignment and comparison statements have been implemented via Expression Evaluator versus the worst performing option, the SELECT against the QSQPABL. Then, try to find a way to eliminate queries against QSQPABL.

I hope this article helps improve the runtimes of your SPL-SPs. I know it has helped some of our customers quite a bit.

Appendix

Indexing and Statistics Strategies for DB2 UDB for iSeries

<http://www-03.ibm.com/servers/enable/site/bi/strategy/strategy.pdf>

OnDemand SQL Performance Analysis Simplified on DB2 for i5/OS in V5R4

<http://www.redbooks.ibm.com/redbooks/pdfs/sg247326.pdf>

Stored Procedures, Triggers, and User-Defined Functions on DB2 Universal Database for iSeries

<http://www.redbooks.ibm.com/redbooks/pdfs/sg246503.pdf>

Centerfield DB2 specific webinars

<http://www.centerfieldtechnology.com/events.asp>

Centerfield newsletter archive

<http://www.centerfieldtechnology.com/publications/>

IBM query optimizer training

<http://www-03.ibm.com/systems/i/software/db2/db2performance.html>

Centerfield consulting services

http://www.centerfieldtechnology.com/services_onsite.asp

**"IBM i is our core expertise.
We promise to deliver
measurable performance
results."**
Mark Holm, CTO



**"Dramatic improvements
immediately"**

**"IBM i is our core expertise.
We promise to deliver
measurable performance
results."**
Mark Holm, CTO



**Emergency performance services
Database performance audits**

**"IBM i is our core expertise.
We promise to deliver
measurable performance
results."**
Mark Holm, CTO



www.centerfieldtechnology.com



Choosing between hardware and software encryption:

An interview with Protegrity Corporation CTO Ulf Mattson

As an increasing number of enterprises seek to protect their data at rest, they are turning to database encryption technologies to help them shelter their assets. However, in choosing between the numerous encryption options in this space they face a dilemma.

Many businesses find themselves grappling with the decision between hardware-based and software-based encryption. Vendors selling database encryption appliances have been vociferously hawking their wares as a faster and more-powerful alternative to software database encryption. Many organizations have bought into this hype based on their experiences with hardware-based network encryption technology.

But database and network encryption are two different animals. Many of the hardware vendor's claims are nothing more than marketing myths, easily refuted by years of evidence to the contrary. Protegrity Corporation CTO Ulf Mattson discusses why software is usually preferable to hardware and how organizations can determine what is right for them.

Let's end the debate: Are hardware-based tools or software-based tools the best way to encrypt and decrypt databases?

I think that might be the wrong question to ask. The right question would be about the topology. What is the right topology to use for database encryption? Remote encryption or local encryption?

The topology is crucial. It will dictate performance, scalability, availability, and other very important factors.

So I think the topic is important but the question is usually not well understood. Usually, hardware-based encryption is remote and software-based encryption is local but it doesn't have anything to do with the form factor itself. Instead, it is about where the encryption is happening relative to your servers processing the database information.

Why do you think people are asking the wrong question?

It is because they are trying to apply what they've learned from other areas of IT. For example, from network encryption they've seen that software doesn't perform as well and that hardware is the best way to accelerate encryption. So they say, "Oh, hardware is the answer, that's the way to offload processing requirements." And they jump to the conclusion that this must be true for database encryption as well.

Why don't these principles apply to database encryption?

When you have, say, credit card data, you have to remember that databases usually operate on the field level. So you cannot send more data than one credit card number at a time. You encrypt it and then you need to give it back to the database immediately because it is sitting there waiting to send the next one. This is particularly a problem with decryption. For example, when someone is searching data and can't wait for slow response times.

When you compare this process between local and remote encryption, the path length—the number of instructions that the computer needs to satisfy your request—is so much longer when you send the data over some kind of network and then receive it back compared to just doing it locally. It could be up to a thousand times longer of a path length to send it to a remote appliance compared to doing the decryption locally on your data server.

Sometimes it is important; in other cases it doesn't really matter.

So how do you determine if it is important to your enterprise?

First, you need to determine whether you are doing batch or online processing.

If it is batch processing you'll need to determine what kind. Are you doing more normal batches where you have a batch job that must run in four hours, or a sequence of batch jobs that need to be finished before morning? Or are you interested in data loading where you can take a chunk of data and go off and process the data and come back and no one is missing it in the meantime?

(Continued on page 13)

If you're doing online transactions then you'll need to determine if it is online transaction processing with transactions that you're processing one record at a time while the user is sitting there waiting, or if you are doing online data warehousing where you're searching large amounts of data at once.

Ok, let's tackle batch processing first. How will the topology affect encryption activities during this type of processing?

If it is a normal batch, I'm sending it over the wire and my database is sitting there and waiting for every credit card number to be encrypted and come back before sending another one and continuing the cycle. If I send it to a remote platform, the batch will stop and wait maybe a thousand times more than I'd expect. So a batch job that would normally take eight hours might take eighty hours. That's not overnight—it isn't even over two nights!

Then we have data loading. Sometimes it can be nice to encrypt the data on a cheaper platform before you load the data into your production database. So you pre-encrypt it, offload it and do it with cheaper equipment. That's provided you have the time.

But if you are, for example, in a retail environment you might have a time constraint. So you need to load the data from your four thousand stores in a certain window. And at that point you may want to use all of the processing power that you have invested in to get the data loaded. That means that you still want to do the encryption locally in your big database server to take advantage of that high end system.

Data loading is the gray area when it comes to local or remote encryption. It depends on whether you really want to offload the processing or if you want to do the data loading very quickly.

And what about online processing?

Both of the major use cases are detrimentally affected by remote encryption.

If I have a data warehouse and the user is sitting there and needs to search among 100 million records or maybe five billion records, like some of our credit card customers, then it's crucial how much time is consumed for each decryption. The person is sitting there waiting for hundreds or millions of records to be decrypted before the answer comes back.

If you do it locally, you may have a response time of around five micro-seconds for each record and then you multiply by 100 million if you have 100 million records and so on. Compare that five micro-seconds for local encryption to the case of remote encryption. You may have a thousand times greater processing time, so if you add up that time the user may wait for an hour instead of one second.

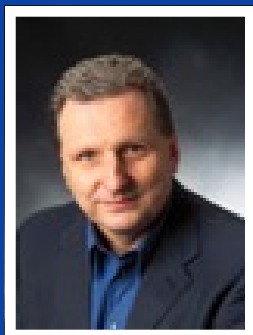
In online transaction processing, one user may not see a difference between local and remote encryption. Because if one user is looking for one record in the database, the difference between five microseconds and five thousand microseconds is not noticeable.

But if you have high volume of processing on your computer it will matter. If you add up all of your transactions and each of them takes a thousand times longer than necessary, you will hit the roof and you will overload your computer. It can really cripple you.

But what if I have a fast network? Won't the speed of the hardware appliances and the speed of the network mitigate the issues you've mentioned?

It is interesting also to notice that fast network doesn't really help you. If you summarize all the steps that need to be processed for the data to go all the way from the database, over to another appliance and back, that path length is so much higher that the network speed doesn't really help you.

Another thing to think about when you dissect this is, if you want to be secure, you actually need to encrypt the wire between the appliance and your database server. Guess what? It costs you more overhead to encrypt that traffic than to do the encryption in the first place.



Ulf T. Mattsson, Chief Technology Officer, Protegrity

Ulf created the initial architecture of Protegrity's database security technology, for which the company owns several key patents. His extensive IT and security industry experience includes 20 years with IBM as a manager of software development, and a consulting resource to IBM's Research and Development organization in the areas of IT Architecture and IT Security.

Ulf holds a degree in electrical engineering from Polhem University, a degree in Finance from University of Stockholm and a master's degree in physics from Chalmers University of Technology.



Bob Cozzi's [RPG World Conference](#) May 7-9, 2009 will be held at the Monte Carlo hotel in Las Vegas. This year, join Bob and the folks that literally wrote the book on System i programming, including Greg Veal, Bruce Hoffman, Bertie Magee, Aaron Bartell and Schadd Gray. At no time has it been more important to attend [RPG World](#).

This year's conference includes the "RPG for the Web" theme. Bob Cozzi is teaching an optional preconference seminar on Wednesday May 6 "RPG for the Web" to get you started using the web browser as the user interface for your RPG IV applications. This seminar is optional and gives those who attend a primer on web technologies and CGI (i.e., "web") programming with RPG IV. Where applicable, "RPG for the web" theme is worked into as many sessions as possible.

The preliminary agenda for the entire conference is online right now, visit www.RPGWorld.com to learn more.

What you'll get at [RPG World](#) that you don't get anywhere else:

- ✓ The best training from the best Instructors.
- ✓ Networking opportunities with our Instructors.
- ✓ Networking opportunities with your Colleagues.
- ✓ Information on the best practices (what works/what doesn't)
- ✓ Information and training that you can take back and use today.

SAVE \$100

Register by March 1, 2009 and save \$100 off the registration fee. Just register with promotion code: **CFTECH**

