Session: BL09

LAB: V5R2 Newest Journal and Commit Features

Larry Youngren
Adam Stallman
Peg Levering

IBM e server iSeries

Trademarks and Disclaimers

IBM *e* server iSeries

IBM @server. For the next generation of e-business.

Contents

Important Note:

If you are not an experienced Journal user and proficient with using the iSeries, you may not be able to complete this entire Lab in the time allotted. Please take a moment to look over the Table of Contents and choose the Labs from which you would benefit most. We suggest completing the Labs in the provided order if you are new to Journaling.

The following are estimates of the time required to complete the various Labs:

Lab 1: Type5 Journal Data	25 to 35 minutes
Lab 2: Journal Standby Mode	25 to 35 minutes
Lab 3: SMAPP Visibility and Tuning	30 to 40 minutes
Lab 4: Savepoints	20 to 30 minutes
Lab 5: Journal Performance Counters and Tracepoints	25 to 35 minutes

Also keep in mind that there are various Optional steps along the way. These steps are not required to complete the Lab.

V5R2 Newest Journal and Commit Features

Lab 1: New Style (TYPE5) Formatting of Journal Data

IBM server iSeries

IBM server. For the next generation of e-business.

Lab 1. TYPE5 Journal Data

In this section of the lab neat, new, never before seen data you can now display by using the **DSPJRN** CL command will be explored.

Introduction

New features in V5R2 allow you to both collect and display more environmental descriptive information with each Journal Entry. Some of this data can be used to better analyze performance while other choices provide more audit or replay information with each Entry. This lab will show you how to enable and view this new data and will demonstrate some ways in which this data can be used.

Objectives

This lab teaches you how to:

- Use the new V5R2 FIXLENDTA option on the CHGJRN command to customize the environment information collected for each Journal Entry.
- Display Journal Entry information using the new V5R2 TYPE5 format including display of Remote Address information as well as the Disk Arm number associated with each Journal Entry.

Lab Information

The notation XX that appears in library names, profile names, and so on, refers to your Team Number (for example, JOTEAMXX, JOLABXX, JODSPXX). Refer to your lab worksheet for details.

Lab Prerequisites

Before you begin this lab, be sure the following prerequisites are available:

- An IBM eServer iSeries or AS/400 with OS/400 V5R2, or higher, with:
  - 5722-QU1 -Query for AS/400
  - 5722-ST1 -DB2 Query Manager and SQL Development Kit for AS/400
  - 5722-SS1 -Feature 5117 (Option 42) AS/400-HA Journal Performance
- The JOLABXX library containing a program and an SQL script for the lab.

Time required

The time required to efficiently complete this lab ranges from 25 to 35 minutes.

Task 1: Introduction to the New Type 5 Journal Data Formatting Option

You have been put in charge of maintaining the vacation records for a large company. This job involves managing vacation records and updating the employee files appropriately to show the remaining number of vacation days. The company has a program in place which verifies the authenticity of the requests and updates the employee records.

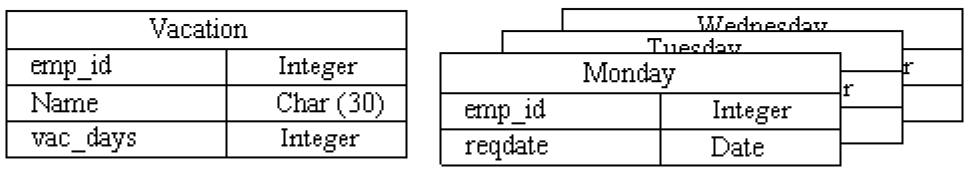
First, you will need to set up a database collection and put data in it to represent these company records.

__ 1. Start interactive SQL with the **STRSQL** CL command.

__ 2. Create your SQL collection with the **Create collection JODSPXX** SQL statement. An SQL collection creates a native library, a Journal named QSQJRN, and various SQL related objects. Database objects created within the collection will automatically be journaled to the Journal QSQJRN. Don't get impatient... creating all of the objects may take a while. There's a lot going on under the covers at this point.

__ 3. Exit interactive SQL with the **F3** option and then select option **1**.

__ 4. We have written a special SQL script for this lab to save you a bunch of keystrokes. Create a set of database tables by executing the SQL statements in the SQLDSP member of the JOLABXX/SQLSTMT file. These tables will automatically be journaled because they are being created in an SQL collection. This can be done with the **RUNSQLSTM SRCFILE(JOLABXX/SQLSTMT) SRCMBR(SQLDSP) DFTRDBCOL(JODSPXX)** CL command. The script executed by this command contains the following SQL statements and will create the database tables shown below:



Vacation	
emp_id	Integer
Name	Char (30)
vac_days	Integer

Monday	
emp_id	Integer
reqdate	Date

Wednesday
Tuesday

Create table Vacation (emp_id int, Name char(30), vac_days int)
Insert into Vacation values (1001, 'Bill', 5)
[Followed by a number of additional employees]
Create table Monday (emp_id int, reqdate date)
[Followed by a number of additional days]
Insert into Monday values (1001, '4/05/02')
[Followed by a number of additional requests]

The Vacation file holds the vacation totals for each employee for the year, revealing the number of days of vacation remaining. The files named after the days of the week contain the pending employee vacation requests for each day which need to be decremented from the Vacation file.

Don't forget to add yourself to the company Database. You wouldn't want to have to do all of this work without receiving any vacation time!

You can accomplish this with the following steps:

__ 5. Start interactive SQL with the **STRSQL** CL command.

__ 6. Enter your own name in Vacation with the following SQL statement:

Insert into JODSPXX/Vacation values(1115, '*your name*', 5)

__ 7. Exit interactive SQL with the **F3** option and then select option **1**.

Now that the data is in place, it is your job to kickoff the program which runs every night to verify the requests and update the company records. This program allows you to specify as input the file containing the new vacation requests for the particular day. Run the program using Monday's requests this time.

__ 8. Set JODSPXX to your current library with the **CHGCURLIB JODSPXX** CL command. This will allow the UPDVAC program used below to access the database objects residing within your collection.

__ 9. Run the JOLABXX/UPDVAC to update the master file with all the new changes made throughout Monday. This can be done with the following CL command: **CALL JOLABXX/UPDVAC PARM('Monday').**

__ Optional: Quickly view the Journal Entries produced so far. This can be done using the **DSPJRN JODSPXX/QSQJRN** CL command. You will see Entries of type PT (Record added) from populating the table, along with UB/UP Entries (Updates) at the end from the UPDVAC program you just ran. Entry #55 should be a PT where your data was added to the Vacation table.

__ 10. Let's route a copy of the Journal Entries which you have produced so far to an OUTFILE which you can then query with the following steps:

__ A. Type **DSPJRN** on the CL command line and press **F4**. Enter Journal, **QSQJRN,** and library, **JODSPXX.** Caution: Don't press enter yet.

__ B. Page down until you find the final keywords, the 'Output' parameter. Type ***OUTFILE** for the 'Output' parameter. Press **Enter**. More options will appear for the 'Outfile format' parameter.

Up until now, everything we've done could have been performed prior to V5R2. Here comes the new stuff...

__ C. Replace *TYPE1 with ***TYPE5** as the 'Outfile format' parameter. This is a new outfile format for V5R2. It allows you to see more of the 'behind the scenes' workings within Journal for all of the available journaled object types so you can make more informed decisions on how to get the most out of your machine.

__ D. Enter **DSPJRNOUT** for 'File to receive output.' Type **JODSPXX** for the library. Press **Enter** twice. You should see a message confirming that 59 Entries were converted. You will see a "Data Truncated to Outfile" message. This is okay. Using the default of *OUTFILFMT for ENTDTALEN limits the Entry specific data written to the outfile to avoid a massively sized outfile.

That's it. We've created an outfile and copied Journal Entries into it.

__ 11. Start interactive SQL with the **STRSQL** CL command. You can now query the outfile we produced above: DSPJRNOUT, which contains the Journal Entries you just copied into this file with the DSPJRN command. To see the myriad of information the new Type 5 formatting option provides, enter: **Select * from JODSPXX/DSPJRNOUT**. Use **F20** (Shift + F8) to scroll to the right and **F19** (Shift + F7) to scroll to the left within the data... Notice all of the column headings.

One of the fields in the table containing the Journal ID (JID) for an object may contain non-displayable characters which will result in an odd looking screen. Press the Escape key if you get stuck due to these non-displayable characters.

If you are an old pro at journaling, you are going to notice some new values on the screen. If you are brand new to Journaling, you may want to ask your Lab instructors to explain some of what you are seeing on the screen.

*You may have noticed some data fields such as 'REMOTE ADDRESS' and 'XID' which have no data in their columns. This is new data available for the first time in V5R2 which is collected at Runtime **only** if you advise the machine to do so... (we didn't). You can use the CHGJRN command and work with the FIXLENDTA* *option to add some of these fields to the information collected for the *TYPE5 display. In the next task, you will try adding some of these options.*

Type5 (rather than the pre-V5R2 styles such as Type4) is the newest kid in town and he's able to show you information about your Journal that others can't. You may want to consider replacing your former use of Type4 with Type5 when you start using V5R2. We'll take a look at how you can use this new information in the upcoming tasks.

__ 12. Exit interactive SQL with the **F3** option and then select option **1**.

Task 2: CHGJRN, FIXLENDTA, and Outfile Querying

In this task we will handle employee requests for the following day (Tuesday), this time gathering and displaying more of the new optional Journal Entry data available for the first time in V5R2.

Your boss has requested more security and performance data to be collected with each Journal Entry. Luckily for you, the new V5R2 FIXLENDTA option is available to help you collect the extra information he's looking for. This information is stored internally in a sub-section of each Journal Entry known as the fixed length area. Hence the name: Fixed_Length_Data.

__ 1. We will need to specify the new FIXLENDTA option on CHGJRN to instruct the machine that the new (optional) Journal Entry information should henceforth be collected.

__ A. Type **CHGJRN JRN(JODSPXX/QSQJRN)** and press **F4** to prompt this command. It would be awfully confusing to represent new data and old-style data in the same Journal Receiver so V5R2 enforces the reasonable rule that you need to change Journal Receivers any time you tinker with the Journal Collection Attributes. Hence, in order to turn on the option to collect the new FIXLENDTA, a new Journal Receiver is required. That's easy... simply specify ***GEN** on the Journal Receiver line instead of the default *SAME.

__ B. Now page down to the 'FIXED LENGTH DATA' (FIXLENDTA) field. Hint: If you prefer to see keywords, you can use F11 to make them appear on your screen. To add more values, type **'+'** on the first line (where it probably says *JOB), blank out the rest of *JOB, and press **Enter**. The 'Specify More Values for Parameter FIXLENDTA' screen should appear. *JOB, *USR, and *PGM are the pre-V5R2 trio of historical "Fixed Length" descriptive information collected for each Journal Entry. This same trio of information can still be collected in V5R2, but they must be specified as three individual options to be used in concert with the newly available Fixed_Length_Data options. Arrow down to a blank line and press **F4**. You will be given a list of various options. These are all of the available parameters for the FIXLENDTA field. Press **F12** to back off this prompt screen and add ***SYSSEQ, *PGMLIB**, and ***RMTADR** into the 'Specify More Values for Parameter FIXLENDTA' screen. Once all of these parameters are set for FIXLENDTA, press **Enter** to accept these values. Now **blank out** the ***JOB** attribute listed as a parameter for FIXLENDTA, and press **Enter** again twice to execute the CHGJRN command.

You will get a message about a Journal sequence number not being reset. Don't worry about it, we didn't specify to reset the Journal sequence number this time.

*Let's review what you have just done: You have selectively enumerated which descriptive bits of information you would like collected for each new Journal Entry produced. Some of the traditional favorites like User_Profile (*USR) and Program_Name (*PGM) were selected, along with some new V5R2 choices, like System_Sequence_Number (*SYSSEQ) and Remote_Addresses (*RMTADR).*

*In this case, we even decided <u>not</u> to collect the Job_Name (*JOB) information that has historically been collected. That was a conscious choice on our part. Harvesting jobs names for each and every Journal Entry costs CPU cycles. By eliminating what we don't really need, we can save those cycles. Of all the traditional pieces of descriptive environmental information collected, the most costly to harvest by far is the program name. The microcode has to climb the execution stack frame by frame looking for the culprit up at the user-level who initiated this Journal request... and that can consume cycles.*

Suppressing collection of program names may be especially attractive if you have a Journal which is predominantly populated by the same batch job executing the same program day after day. Do you really need to have all 100,000 Journal Entries look up the program name anew? If not, why not suppress it? Hmm... there is a good tip you can take home. If you don't have a need for the information, you can help performance and save space in the Journal Receiver by not collecting the information! That's the beauty of the new "Fixed Length Data" versatility V5R2 affords. (Got any spurious Journal overhead you can eliminate when you return home?)

__ 2. If you didn't already do this in Task 1, set JODSPXX to your current library with the **CHGCURLIB JODSPXX** CL command.

__ 3. Run the JOLABXX/UPDVAC to update our master vacation file with all changes made throughout Tuesday. This can be done with the following CL command: **CALL JOLABXX/UPDVAC PARM('Tuesday').**

__ 4. Let's route a copy of the new Journal Entries which you just produced to an OUTFILE by executing the following command: **DSPJRN JRN(JODSPXX/QSQJRN) OUTPUT(*OUTFILE) OUTFILFMT(*TYPE5) OUTFILE(JODSPXX/DSPJRNOUT)**. This will overwrite the same table which you created in step 10-D of the previous task by prompting on the DSPJRN except that it is done all in one step. You should see a confirmation message pointing out that 9 new Journal Entries were converted.

Next you will be selectively querying and displaying portions of the resulting output file, so you will need to know the field names to employ in the SQL SELECT statement which appears below.

__ 5. Enter **DSPFFD JODSPXX/DSPJRNOUT**. To view the field names of all of the fields in a file, you can use the DSPFFD (display file field description) CL command. Page down and you will be able to see the field names corresponding to each field a *TYPE5 formatted Journal Entry houses, such as: JOSYSSEQ, JOPGMLIB, JOUSER, and JOPGM. Exit using **F3**.

*You may be asking yourself, "Why do I need all of this <u>extra</u> data? What can I do with it?" In fact, this extra data shows you many things about your Entries which you might find useful. For example, you may have many programs at home with the same name, but residing in different libraries. By including the new V5R2 *PGMLIB data, you can find out exactly which instance of the program made the changes to the database record and where this program can be found. You can see this in action right now. Everyone in this lab is running a program, each having the <u>same</u> name, UPDVAC, yet in their own individual libraries. When you display the Entries of your Journal, you will see that the UPDVAC program which you ran was from your own library (JODSPXX) and not from someone else's library. Let's take a look at those Entries...*

__ 6. Start interactive SQL with the **STRSQL** CL command.

This time, instead of viewing all of the fields present in the output file for each Journal Entry, we'll view only the customized FIXLENDTA ones you have just included using a refined SELECT.

__ 7. Enter **Select JOSEQN, JOSYSSEQ, JOPGMLIB, JOUSPF, JOPGM, JORADR from JODSPXX/DSPJRNOUT**. Now only those six fields will be shown, and the fields (like JORADR) which did not previously have any data when we performed Monday night's refresh (because we had not yet instructed the Journal to start collecting such information) will now have Tuesday's information in their columns rather than blanks. If you don't see all six column headings, you probably left out some commas. Exit this display using **F3**.

You have now seen how the FIXLENDTA option allows you to collect exactly the customized information you prefer with each Journal Entry. No more, and no less. This feature provides you with the ability to only use the processing power and disk space to collect what is needed in your own shop. Once you have determined what information you need, you'll probably want to customize your own Journals so as to discard the chaff.

The Security Audit Journal on a V5R2 system will include all of the Fixed Length Data information that is possible to collect. The Journal Entries produced in this Journal will now have this extra Audit information available.

So let's review:

1) *You can use the new V5R2 CHGJRN options to customize what is collected.*
2) *You can then use the new V5R2 DSPJRN *TYPE5 to customize what is displayed.*

Task 3: Intrusion Detection

Task 3 demonstrates the ability to see the IP address of the remote origin machine associated with a given Journal Entry using the new V5R2 Type5 Journal Display Option. It is especially useful if you allow remote access to your machine.

Now that you have the hang of the vacation request program, wouldn't it be great to just go in and give yourself a few more vacation days? The profile you are using is shared by a number of developers, so there is no way for the auditors to figure out who did it, right? Let's give it a try and see what happens!

__ 1. Without using the standard UPDVAC program, let's sneak in the backdoor and use native SQL to update your own data record to have more vacation days.

Update JODSPXX/Vacation set vac_days = 30 where emp_id = 1115

__ 2. Exit interactive SQL with the **F3** option and then select option **1.**

That SQL update undoubtedly produced a Journal Entry. Now that a new Journal Entry has been produced in your Journal Receiver, let's refresh the contents of our outfile so we can query what happened.

__ 3. Type **DSPJRN JRN(JODSPXX/QSQJRN) OUTPUT(*OUTFILE) OUTFILFMT(*TYPE5) OUTFILE(JODSPXX/DSPJRNOUT)** on the CL command line and run the command (you may be able to use F9 to retrieve this command since you used it in the last task). You should see a confirmation message that 11 Entries were converted.

__ Optional: DSPJRN to screen and look at the 'Job' column. **DSPJRN JRN(JODSPXX/QSQJRN)**. You will see that no Job Name was recorded. Rather, such information was '*OMITTED.'

Feeling smug, aren't you? Maybe nobody can detect what you did.

*Do you remember when we added *RMTADR to the FIXLENDTA with the CHGJRN command back in step 1-B of task two? Well, here is where you can use that remote address data we've been collecting.*

__ 4. Start interactive SQL with the **STRSQL** CL command.

__ 5. Enter **Select JOSEQN, JOPGM, JORADR, JORPORT, JOUSPF from JODSPXX/DSPJRNOUT**. Find the last Journal Entry. That's the one that corresponds to the change you recently made to your number of vacation days. Use F20 (Shift + F8) to scroll to the right to view the entry. Notice that the PROGRAM NAME field indicates that it was not the authorized UPDVAC program which made the change to the Vacation file! It was you typing a command. Also note that the REMOTE ADDRESS field of the Entry indicates where the change originated from. This shows the IP address of the clandestine remote computer making the request!

Record the IP address here: ________________

__ 6. Exit interactive SQL with the **F3** option and then select option **1**.

__ 7. Find the IP address of the machine you are currently working on. Open a simple Command Prompt by clicking on the Microsoft Windows Start Button, choosing **'RUN',** and entering **'command'** before clicking **'OK'**. Type **'ipconfig'** and press **Enter**. Some information will be displayed, including your PC's IP address.

Record the IP address here: ________________

Type **Exit** and press **Enter** to exit the Command Prompt.

Your machine's IP address is the same as the one posted in the 'Remote Address' field of the outfile. BUSTED! Maybe giving yourself a few extra days wasn't the best idea! The boss seems to be posting a help-wanted sign in the window. Looks like you're going to have a loooooong vacation now. The new Journal Entry data can help to pinpoint exactly how a particular change was made (and by whom and from where!).

Hmm... your auditors are going to love this option.

Task 4: Compare Disk Arm Behavior when Using Journal Caching

Okay, now you see how that works. Why don't we take a peek at another item that has just become available with V5R2: the identity of the disk ARM housing each Journal Entry. This field tells you which arm (or disk drive) a certain Journal Entry resides on. This can be used to determine the "bundling rate" and scattered disk write pattern of Journal Entries. Armed with this information you can determine how many Journal Entries are being written to disk together at the same time. And why would you want to know this? The more Journal Entries in a bundle, the fewer disk writes that need to be done. The Journal Caching option (available for the first time as a native Journal Attribute in V5R2) is one way to achieve enhanced bundling.

In the Appendix you can find the "BUNDLE" program which you can use to help reduce the row disk arm data and calculate the average bundling rate.

Let's give this stuff a try.

__ 1. Turn on the V5R2 Batch Journal Caching feature for your Journal by calling the CL command provided by the Journal Caching BOSS Option (we pre-loaded this new option on our lab machine). **CHGJRN JRN(JODSPXX/QSQJRN) JRNCACHE(*YES).** This new V5R2 Boss Option is available for a fee and can be ordered as 5722-SS1 Option 42.

You may have noticed that we didn't have to generate and attach a new Journal Receiver in order to execute this command. In fact, the caching feature can be toggled on or off as often as you wish. Some shops will want it on during Batch and off during the interactive portion of the day.

__ 2. Set JODSPXX to your current library with the **CHGCURLIB JODSPXX** CL command. The Batch Journal Caching feature's mission in life is to cache multiple Journal Entries in main memory, then write a bunch of them in unison to the same disk arm in one disk revolution for the sake of efficiency.

__ 3. Run the JOLABXX/UPDVAC to update all changes made on Wednesday. This can be done with the following CL command: **CALL JOLABXX/UPDVAC PARM('Wednesday').**

Let's see what the Journal can tell us regarding disk arm usage.

__ *4.* Type **DSPJRN JRN(JODSPXX/QSQJRN) OUTPUT(*OUTFILE) OUTFILFMT(*TYPE5) OUTFILE(JODSPXX/DSPJRNOUT)** on the CL command line and run the command (you may be able to use F9 to retrieve this command since you used it in the last task). You should see a confirmation message showing that 40 Journal Entries were converted.

*You will recall that *TYPE5 is the new V5R2 stuff that let us take a peek at formerly hidden information (such as arm number). In past releases, only microcoders and service personnel in IBM's laboratories could see this information. With V5R2, you can too! In Task 2 we worked with some of the customized FIXLENDTA types, which are not normally collected, but must be 'turned on' if you want to collect and view them. However, certain information (such as arm number) have always been collected and have only become available for viewing in V5R2.*

__ 5. Start interactive SQL with the **STRSQL** CL command. We performed Monday's and Tuesday's nightly refreshes without the benefit of the Journal Caching option. So those Journal Entries should be scattered all over the disk arms in dribs and drabs. You can now query the outfile DSPJRNOUT to see if the dispersion of data onto the arms is different for Wednesday's refresh, since it had the benefit of Journal Caching.

The identity of the disk arm to which each Journal Entry was written is revealed in the JOARM column of our Type5 outfile.

__ 6. Enter **Select JOSEQN, JOARM, JOENTT from JODSPXX/DSPJRNOUT** to view the sequence numbers, arm numbers, and Entry types of the Journal Entries you have produced.

You will notice that the Journal Entries before the CI Journal Entry (marking the transfer into Caching mode) are spread out over the arms, with only a couple of Entries in a row going out to the same arm at one time. This modest bundling is because the update before and after images (UB and UP) are always bundled together. Hence Monday and Tuesday's runs were not disk efficient.

Next, compare this to the improved bundling that occurs when you use a feature such as Caching. After the CI Entry, nearly all of the Journal Entries are bundled onto one arm and are written to disk with a single disk write. (Interactive SQL closes the file after your first update forcing the first set of entries to their own arm.) If enough Journal Entries had been produced to completely fill the Journal buffer, these Entries would have been written out to disk and then the next arm would have been used for a second bundle. The Entries are bundled on the arms because they have been 'saved up' before being journaled, thus many Entries are being written to disk at once. Fewer disk writes means much better performance for Wednesday's refresh!

Want to be a hero when you return home? Why not propose speeding up your nightly Batch Jobs by turning on this V5R2 Caching feature?

How will you know if you need it? ...Display your arm numbers!

__ 7. Exit interactive SQL with the **F3** option and then select option **1**.

*Various factors including commitment control, an OVRDBF with SEQONLY *YES, Journal Caching, and high levels of concurrent processing from multiple jobs affecting the same Journal can improve the bundling rate in your own environment. If you currently have a low bundling rate, it is likely that the usage of these features can greatly increase your Journal performance. With V5R2 and the *TYPE5 option you now have the tools to analyze disk efficiency of journaling in your own shop.*

*For more information regarding Journal Bundling and Cache tuning and the associated performance benefits you may want to take a look at the Redbook: '**Striving for Optimal Journal Performance'** found on the IBM Redbook Web site: www.redbooks.ibm.com*

IBM

V5R2 Newest Journal and Commit Features

Lab 2: Journal Standby Mode

IBM eserver iSeries

Lab 2. Journal Standby Mode

In this section of the lab the Journal **Standby** feature will be explored. Standby is a new V5R2 feature available only if you have installed Option 42 of OS/400. It makes the most sense for shops that have a high availability requirement.

Introduction

The main purpose of the Journal Standby feature is to speed up the process of switching over from a production system to a hot backup system. Standby mode allows this to be done without hurting run time on a backup system and without having to endure time-consuming Journal enabling steps during switching over. It allows you to prepay your start-up costs well in advance.

In short: Standby is what you do on your <u>target</u> machine.

Objectives

This lab teaches you how to:

- Enable Standby mode for a Journal.
- Compare the performance between normal journaling, Journal Caching, and Journal Standby mode.
- Compare the performance between changing from Standby mode to active journaling vs. starting journaling for your objects.
- Put out critical Journal Entries to a Journal despite the fact that it's in Standby mode.

Lab Information

The notation XX that appears in library names, profile names, and so on, refers to your Team Number (for example, JOTEAMXX, JOLABXX, JOSBY_A_XX). Refer to your lab worksheet for details.

Optional steps are included in this lab in addition to the required steps. These steps are not required to complete the lab, but can be attempted to further demonstrate the usage of various Journal commands. Optional steps are denoted with "Optional:" instead of the step number.

Lab Prerequisites

Before you begin this lab, be sure the following prerequisites are available:

- An IBM eServer iSeries or AS/400 with OS/400 V5R2, or higher, with:
  - 5722-QU1 -Query for AS/400
  - 5722-ST1 -DB2 Query Manager and SQL Development Kit for AS/400
  - 5722-SS1 -Feature 5117 (Option 42) AS/400-HA Journal Performance
- The JOLABXX library contains a program and an SQL script for the lab.

Time required

The time required to efficiently complete this lab ranges from 25 to 35 minutes.

Task 1: Producing the Performance Baseline

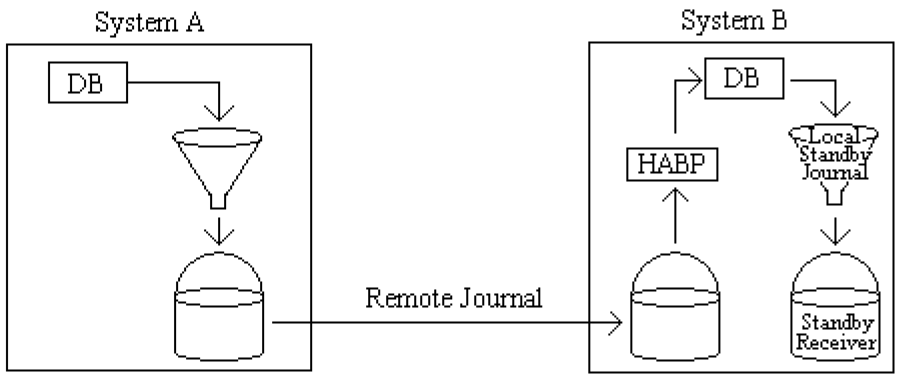
Your company has a large database system and you are in charge of keeping it running, and bringing it back up if the system goes down. You have a main system A which registers all of your company's transactions. These transactions are then remote journaled and replayed on another system, B. Should system A go down, system B must take over your production workload. (See the diagram on the next page)

Before the new Standby feature in V5R2, you had only two choices: 1) you would either always journal your entire replicated database and other journaled objects on system B or 2) wait and start journaling these objects only when you want to switch over to this system.

Both of those former choices had disadvantages:

- The First Alternative: Always journaling your entire replicated database and objects on the target system takes up resources day after day after day... and may slow down the replication process itself taking place on system B. In fact, if you're using HABP (High Availability Business Partner) software to replay database and object changes to the redirected objects residing on this system, this option would have increased the likelihood that such HABP jobs would have a tough time keeping up. (Ever seen your HABP replay jobs begin to fall behind? This could be one of the causes!) By contrast, with the new Standby feature turned on, you will not be consuming as many valuable system resources on the target system so you will not incur as much performance impact.

- The Second Alternative: Although starting journaling for a simple environment on a few objects at the last minute on your target system, during a so-called role-swap, from system A to system B may not take long, starting journaling on thousands of objects on the target system will take a very long time. If you have an environment consisting of hundreds of objects, that's the risk you face. Are you willing to make your customers (and your impatient boss) wait for all of this activity to complete before a switch from system A to system B can be completed? Probably not. With the new Standby feature in V5R2, you won't have to. If your primary machine goes down and you have your secondary machine set up such that your critical objects are already journaled to a Journal which is in Standby mode, you can start running on system B very quickly by simply taking the Journal out of Standby mode. It's that simple! Task 4 allows us to find out how rapidly we can make that transition. (Trust us, you're going to be impressed.)



You look like an entrepreneur with a sweet tooth who'd like to open a gourmet cheesecake factory. Have we got a deal for you....

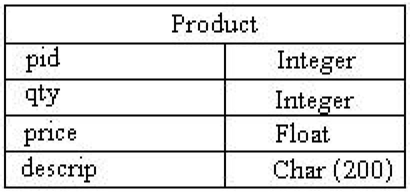
In this section of the lab you will create a set of database tables and indexes representing a cheesecake factory. Sounds yummy, doesn't it? This environment will then be used to establish a baseline measurement for comparison against a similar environment with Journal performance enhancements such as Standby mode.

__ 1. Start interactive SQL with the **STRSQL** CL command.

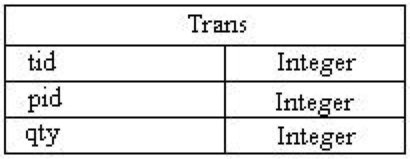
__ 2. Create our first SQL collection (we'll call it collection 'A') with the **Create collection JOSBY_A_XX** SQL statement. An SQL collection creates a native library, a Journal named QSQJRN, and various SQL related objects. Database objects created within collection 'A' will automatically be journaled to the Journal QSQJRN. Give this some time to complete... There are a lot of underlying objects to create.

__ 3. Exit interactive SQL with the **F3** option and then select option **1**.

__ 4. Create a set of database tables and indexes by executing the SQL statements in the SQLSBYPERF member of the JOLABXX/SQLSTMT file. This can be done with the **RUNSQLSTM SRCFILE(JOLABXX/SQLSTMT) SRCMBR(SQLSBYPERF) DFTRDBCOL(JOSBY_A_XX)** CL command. The script executed by this command contains the following SQL statements and will create the database tables shown below:



Product	
pid	Integer
qty	Integer
price	Float
descrip	Char (200)



Trans	
tid	Integer
pid	Integer
qty	Integer

```
Create table Product (pid int, qty int, price float, descrip char(200))
Create table Trans (tid int,  pid int, qty int)
Create index prod_pid on Product (pid)
Create index trans_ids on Trans (pid, tid)
Insert into Product values (1, 200, 12.00, ‘Cherry Cheesecake’)
Insert into Product values (2, 100, 10.00, ‘Lemon Cheesecake’)
[ Followed by a number of additional products ]
```

You may see brief progress messages flash across the bottom of your screen as Access Paths are built.

__ 5. Set JOSBY_A_XX to your current library with the **CHGCURLIB JOSBY_A_XX** CL command. This will allow the PERFRUN program used below to access the database objects residing within your collection.

At this point your Transaction table (Trans) is empty (i.e. no one has placed any orders yet). We’re about to create some transactions to fill this table, so hold onto your hat! Our freezer is full, all of the school groups are starting their fundraisers, and we’re about to start shipping product.

__ 6. Run the JOLABXX/PERFRUN program for 30 seconds. This can be done with the following CL command: **CALL JOLABXX/PERFRUN PARM(‘30’).** This program will insert records into the Transaction table and update the Quantity field in the Product table thereby simulating the shipments we’ve made. The program will terminate after running for 30 seconds. After 30 seconds have passed and the completion message appears on your screen, press **Enter** to get back to original session.

Our purpose is to determine how many transactions we can service in 30 seconds. This will be our baseline for comparison.

__ Optional: The Journal Entries that have been produced during this 30 second run can be viewed with the **DSPJRN JRN(JOSBY_A_XX/QSQJRN)** CL command. After the CT (Create Table), MC (Member Create), and JF Journal File) Entries which put our tables in place, you should see a series of Journal Entries such as PT (Put in a Transaction) and UP (Update our inventory) which were generated by the PERFSBYRUN Program. You may have to page down to see them all. Enter 5 next to Entries to browse through some of the UP flavored Entries from the Product table so you can see the kinds of merchandise your hard earned dollars have bought.

Gosh, it looks like a good day - turtle cheesecakes and cheesecake minis are selling well!

__ 7. Again start interactive SQL with the **STRSQL** CL command.

__ 8. Determine how many transactions were produced during this run of the performance program with the **Select count(*) from JOSBY_A_XX/Trans** SQL statement.

Record the number of transactions completed here: ________

Your new business has a profit margin of $1.00 on each cheesecake you sell. Hmm... nice profit for 30 seconds of work.

The faster your Journal environment ran and the smaller the Journal overhead, the greater the number of complete transactions that will have been produced in 30 seconds. This value will serve as your performance baseline. This is also the value we're going to try to beat by tuning our Journal environment. It represents the highest sustained rate of cheesecake sales your application can muster. The question is: "Can we do better?"

Task 2: Running with Batch Journal Caching

Task 2 illustrates the performance improvement we can achieve by enabling the V5R2 Batch Journal Caching feature. The impact of this performance enhancement will depend greatly upon the specific environment being used.

Before Standby mode arrived in V5R2, turning on Caching via the Journal Caching PRPQ on the backup system was the best software option to reduce the overhead of journaling on your target machine. Let's see how Caching compares to our baseline and later we'll give Standby mode a try.

__ 1. Start interactive SQL with the **STRSQL** CL command if you are not already in interactive SQL from the previous task.

OK, it's time to create collection 'B' in order to do a comparison.

__ 2. Create an SQL collection with the **Create collection JOSBY_B_XX** SQL statement. An SQL collection creates a native library, a Journal named QSQJRN, and various SQL related objects. Database objects created within collection 'B' will automatically be journaled to the Journal QSQJRN. Yes, yes... this step may take a little while, but you know that.

__ 3. Exit interactive SQL with the **F3** option and then select option **1**.

__ 4. Create your tables and indexes by running the SQL statements in the SQLSBYPERF member of the JOLABXX/SQLSTMT file. The can be done with the **RUNSQLSTM SRCFILE(JOLABXX/SQLSTMT) SRCMBR(SQLSBYPERF) DFTRDBCOL(JOSBY_B_XX)** CL command. The script contains the same SQL statements we used in task 1 and will create the same database tables shown in task 1, but in collection 'B' this time.

__ 5. Turn on the new Batch Journal Caching feature for the Journal in collection 'B' by calling the CL command using the new V5R2 option enabled by the HA Journal Performance BOSS Option with the following command: **CHGJRN JRN(JOSBY_B_XX/QSQJRN) JRNCACHE(*YES).** This option is available for a fee (and worth every penny) and can be ordered for V5R2 as 5722-SS1 Option 42.

__ Optional: To confirm that the Journal knows Caching is enabled, let's do a WRKJRNA to display the Journal attributes. **WRKJRNA JRN(JOSBY_B_XX/QSQJRN)**. Also note that *MAXOPT2 has become the new default setting for SQL collections in V5R2.

Those of you who've used similar support in the form of the 'Journal Caching PRPQ' for release V4R4, V4R5, or V5R1 will recognize this new V5R2 choice. The former PRPQ has grown up! It's now available as a simple keyword on a regular Journal command and its scope has been expanded to Cache not just database changes, but also journaled IFS, Data Area, and Data Queue changes as well.

Let's get ready for our head to head performance comparison:

__ 6. Set JOSBY_B_XX to your current library with the **CHGCURLIB JOSBY_B_XX** CL command. This will allow the PERFSBYRUN program to run against the tables residing in your 'B' collection.

__ 7. Run the JOLABXX/PERFSBYRUN program for 30 seconds. This can be done with the following CL command: **CALL JOLABXX/PERFRUN PARM('30').** This program will insert records into the Transaction table and update the Quantity field in the Product table. The program will terminate after running for 30 seconds. Press **Enter** get back to original session.

__ Optional: The Journal Entries that have been produced can be viewed with the **DSPJRN JRN(JOSBY_B_XX/QSQJRN)** CL command.

__ 8. Again, start interactive SQL with the **STRSQL** CL command.

__ 9. Run the following SQL statement to determine how many transactions were produced during this Cache-enabled run of the PERFSBYRUN program.
Select count(*) from JOSBY_B_XX/Trans.

Record the number of Transactions completed here ______

How does that compare to the value you recorded for step 8 of task 1? You accomplished more work in 30 seconds this time, right?

As you can see, the Journal Caching feature has a large beneficial impact on the performance of your journaling environment. The Journal Caching feature has been made available on pre-V5R2 machines as an optional, nonstandard part of OS/400. It is available for V4R4, V4R5, and V5R1 through the Batch Journal Caching PRPQ 5722-BJC plus a matching set of PTFs. It is available for V5R2 as OS/400 optional feature 42. Both the pre-V5R2 version and the V5R2 optional feature version have a FREE trial version! That's right, you can try out these fabulous options on your own system without spending a dime.

You can read more about this PRPQ and the benefits of Journal Caching in the Redbook: "**Striving for Optimal Journal Performance**" which can be found at: www.redbooks.ibm.com.

Wow, this sure does reduce the journaling overhead! This option should allow for those replay jobs to more easily keep your hot backup in sync with your production system. Hmm... at $1.00 profit per cheesecake, you're getting filthy rich, right?

Task 3: Running with Journal Standby Mode

The question is: "Good though Caching is, can we do even better?"

Task 3 shows the performance improvement when using the Journal Standby option alone without any other performance improvements. The impact of this performance enhancement will depend greatly upon the specific environment being used.

__ 1. Start interactive SQL with the **STRSQL** CL command if you are not already in interactive SQL from the previous task.

OK, it's time to create collection 'C' in order to do yet another comparison.

__ 2. Create an SQL collection with the **Create collection JOSBY_C_XX** SQL statement. An SQL collection creates a native library, a Journal named QSQJRN, and various SQL related objects. Database objects created within collection 'C' will automatically be journaled to the Journal QSQJRN.

__ 3. Exit interactive SQL with the **F3** option and then select option **1**.

__ 4. Here comes the critical new piece for V5R2: turn on the new Standby feature for the Journal in collection 'C' by calling the CHGJRN CL command: **CHGJRN JRN(JOSBY_C_XX/QSQJRN) JRNSTATE(*STANDBY).**

__ Optional: Confirm the fact that our journal is now in Standby mode by using WRKJRNA. **WRKJRNA JRN(JOSBY_C_XX/QSQJRN)**. You should see *STANDBY beside 'Journal state.'

The Journal is now in Standby mode. Gosh, that was easy! The journaled objects are no longer being actively protected and thus the normal Journal Entries will <u>not</u> be produced nor written to disk. That's the whole point of Standby mode. If used properly, such as on our hot backup system where we do not require this protection until we switch over, Standby mode can provide a great performance boost... Let's find out how much:

__ 5. Create your tables and indexes by running the SQL statements in the SQLPERF member of the JOLABXX/SQLSTMT file. The can be done with the **RUNSQLSTM SRCFILE(JOLABXX/SQLSTMT) SRCMBR(SQLSBYPERF) DFTRDBCOL(JOSBY_C_XX) COMMIT(*NONE)** CL command. The script contains the same SQL statements we used in task 1 and will create the same database tables shown in task 1.

Note: Commit(*NONE) is used here because commitment control is not valid with Standby mode. Commitment control transactions would not be able to be rolled back without the Journal Entries being recorded in the Journal. If your application makes use of commitment control, your best option on the Target system will be to use Journal Caching.

__ 6. Set JOSBY_C_XX to your current library with the **CHGCURLIB JOSBY_C_XX** CL command. This will allow the PERFSBYRUN program to run against the SQL tables residing within your 'C' collection.

__ 7. Run the JOLABXX/PERFSBYRUN program for 30 seconds. This can be done with the following CL command: **CALL JOLABXX/PERFRUN PARM('30').** This program will insert records into the Transaction table and update the Quantity field in the Product table. The program will terminate after running for 30 seconds. Press **Enter** get back to original session.

__ 8. Again, start interactive SQL with the **STRSQL** CL command.

What do you think? Will Standby really turn out to be even more efficient than Caching mode?

__ 9. Run the following SQL statement to determine how many transactions were produced during this run of the PERFSBYRUN program. **Select count(*) from JOSBY_C_XX/Trans.**

Record the number of Transactions completed here ______

As you can see, the Journal Standby feature can have a large positive impact on the performance of a Journaling environment. When the Journal is in Standby mode, almost nothing is being written out to the Journal or the related Journal disks, thus object changes can be replayed to your replica environment more quickly. This will allow your backup system to better achieve the objective of remaining in "lock-step" with your source system, just in case you need to switch over to the backup... And what would you do after such a role swap? Well... you'd issue a CHGJRN command so as to leave Standby mode and re-enable ordinary full journaling support.

How many more transactions did you produce during the Standby run in Step 9 of Task 3 than during the original baseline of Task 1, Step 8? See... we told you, you would be impressed.

__ 10. Exit interactive SQL with the **F3** option and then select option **1**.

Why not compare the Journal Entries from our baseline run with those produced during our Standby Journal run?

__ 11. Enter **DSPJRN JOSBY_A_XX/QSQJRN.** Page down to the final screen. Use **F3** to exit when you are ready to move on.

Whoa! There are a lot of Entries for our original Baseline in collection 'A'! Every change to your rows within the SQL tables have been logged to the Journal! Do you wonder what the Standby Journal looks like? Why don't we go take a look?

__ 12. Enter **DSPJRN JOSBY_C_XX/QSQJRN.** Exit using an **F3** when you are ready.

Quite a difference, isn't it?

For collection 'C,' none of the row by row database changes were Journaled! No wonder it was so fast. Yet, that's exactly what we told the database to do by enabling Standby mode. Standby mode has the additional benefit of not using up disk space on your backup system or contending for disk resources on the system.

So let's review: You'd use Standby on the target system, not the production system. You'd leave Standby mode only when a role swap ensues, making your former target system take on a production role. And why would this be so attractive? ...Because we don't want to incur the full overload of starting journaling at role swap time... especially if our factory has thousands of journaled objects.

So... how much time would we actually save at role-swap time by our insightful use of Journal Standby mode? Let's find out...

Task 4: STRJRN vs. Standby --> Active Transition

Let's try a little comparison to test the validity of the idea that it is faster to keep your Journals in Standby mode and change them to Active mode at role swap time than it is to start journaling on all of your objects at role-swap time. This is exactly what you would need to do in the event you need to switch over to your backup system.

You have decided to employ this new V5R2 Standby feature as the role-swap strategy in your factory, but your pesky, know-it-all brother-in-law insists on employing the older vintage strategy of starting journaling for all the files in his own factory at roll swap time. He's always done it that way, and hence believes there is no reason to change his practices for V5R2. (He also still wears bell bottoms...) You both use the same application package so this is a true "apples to apples" comparison. Oops! Both of your production systems went down at the same time (must be that new so-called killer 'fix' you installed last night)! You make a bet to see who gets their machine up and running first!

Remember, your brother-in-law must start journaling from scratch on all of his files residing on the target system. You can see these files in library JOFILEXX. Let's take a peek.

__ 1. Type **DSPLIB JOFILEXX** to see the files that will need to have journaling started. In the upper right hand corner of the display the 'Number of objects' is displayed.

Record the number of files here: ________________

Exit back to the Command Entry Screen with an **F3**.

How does this compare to the quantity of files in your shop? Pretty timid, huh? But probably sufficient to make our point. It's time to start simulating your brother-in-law's experience.

__ 2. Type **JOLABXX/JOTIMEIT (CALL JOLABXX/STRJRNLIB PARM('JOFILEXX' 'QSQJRN' 'JOSBY_A_XX'))** on the CL command line. The STRJRNLIB program will start journaling on all of the files in JOFILEXX. This is going to take a while, so be patient. By using the JOTIMEIT command, the time it takes to do this will be returned.

Record the it took to start journaling here: ________

__ Optional: You can use **F10** to see detailed messages and thus view the list of files for which journaling was started.

That took a while, didn't it? Your brother-in-law must be a patient man. I wonder if he is still confident that he will win? (Hope you bet a bundle on the outcome of this test).

What if there were one thousand more files? Ten thousand? One hundred thousand? The time could really add up. This isn't just normal "go grab a Coke" time... this is when your boss is standing next to you waiting for your backup system to complete the role-swap after your production system went down!
Some popular ERP packages in the marketplace place tens of thousands of files on your system. Can you imagine what role-swap time must be for such packages if Standby journaling isn't used?

Let's reuse the environment we just created and this time put our Journal into Standby mode.

__ 3. Type **CHGJRN JRN(JOSBY_A_XX/QSQJRN) JRNSTATE(*STANDBY)** on the CL command line.

__ Optional: Confirm the fact that our journal is now in Standby mode by using WRKJRNA. **WRKJRNA JRN(JOSBY_A_XX/QSQJRN)**. You should see *STANDBY beside 'Journal state.'

That was simple. In fact, that's all it takes on your target machine to put Standby mode in place.

Okay, now it's your turn! We know how long it took your brother-in-law's system to complete the journaling phase of the role. Let's see how long it takes to bring your Journal back to Active mode from Standby. Will there really be an improvement? Will you beat your brother-in-law in the race to get your target system back up first? We'll soon find out!

__ 4. Type **JOLABXX/JOTIMEIT (CHGJRN JRN(JOSBY_A_XX/QSQJRN) JRNSTATE(*ACTIVE))** on the CL command line. This will return the time it takes to change <u>all</u> of the journaled files from Standby mode to Active mode.

Record the time it took to make the transition to Active mode here: _____

Whoa! That is quite the difference in time! Imagine the difference this simple switch would make over the STRJRN method if you had many, many more objects. Your customers (and your boss) will be happy that you could make the system available again so quickly. AND, you won the bet.

__ Optional: Let's confirm that the Journal really did make the transition back into Active mode. To do this, we will use the WRKJRNA command. **WRKJRNA JRN(JOSBY_A_XX/QSQJRN)**. You should now see *ACTIVE, rather than *STANDBY, beside 'Journal state.'

So... let's summarize: If you want your day to day overhead on the target system to be ultra low prior to V5R2, you had to leave journaling turned off, but that makes the actual role-swap slow. Yuk! V5R2 changes all of that. You can enjoy <u>both</u> day to day low overhead HABP replay on the target machine <u>and</u> fast role-swaps. How? Merely employ Standby mode on your target machine. Sounds like you and your boss need to have a discussion regarding these new V5R2 choices when you get home.

And how do you get your hands on this impressive performance boost? Why Option 42 of OS/400, of course.

Optional Task 5: Sending Entries to a Journal in Standby Mode

In this section of the lab you will explore some of the critical operations that will be journaled regardless of the system being in Standby mode. These operations include object deletes, renames, restores, and starting and stopping journaling transactions, along with a number of others. These types of uncommon Journal Entries will flow out to the Journal Receiver even though the Journal is in Standby mode. Why? Because the Entries are uncommon enough to not cause a performance impact and some are essential to the proper recovery of the machine. (You wouldn't, for example, want to lose a file rename, would you?)

In this task you will also be introduced to a new SNDJRNE option which is provided to allow you to briefly override the Standby setting for selected critical application steps. The SNDJRNE option is useful if you only want a few, specific application-provided Journal Entries to be written out.

You will be working with the JOSBY_C_XX library. You previously placed it in Standby mode back in step 4 of task 3.

__ Optional: Why don't we make sure that the Journal really is in Standby mode? To do this, we will use the WRKJRNA command. **WRKJRNA JOSBY_C_XX/QSQJRN**. You should see *STANDBY beside 'Journal state.'

One type of Journal Entry that is journaled regardless of Standby mode is an object rename operation. Let's perform a rename on a file and see what happens in our Standby Journal.

__ 1. Rename a file in JOSBY_C_XX. **RNMOBJ OBJ(JOSBY_C_XX/TRANS) OBJTYPE(*FILE) NEWOBJ(SALES).**

Now display the Journal to see which Entries have been produced despite the Journal being in Standby mode.

__ 2. Type **DSPJRN JOSBY_C_XX/QSQJRN.**

You will see two Entries in the Journal, FN (File Renamed) and MN (Member Renamed). As you can see, even though Standby mode is on, these essential DDL Entries were written to the Journal.

That's the good news. The system honors Standby mode for plain-Jane Journal Entries like PT, UP, DL which tend to be numerous, but is smart enough to recognize critical DDL operations like FN, MN and let them reach the Journal. And what happens to the PT's and UP's? They're tossed into the proverbial "Bit Bucket" and vanish into thin air.

Many applications write out their own application-initiated Journal Entries to a Journal during processing for a variety of reasons. If you have an application which uses the SNDJRNE CL command or the QJOSJRNE API to do this and your Entries are so critical that you don't want to risk having them dropped into the bit bucket, you may want to make use of a new option which will allow Journal Entries to flow out to the Journal even though it is in Standby mode.

Let's see how this works:

__ 3. We'll use the SNDJRNE command to send a new Journal Entry. Don't press Enter until you have completed filling in all of the fields required.

__ A. Type **SNDJRNE** and press **F4** to prompt on this command.

__ B. Enter Journal, **QSQJRN,** and library, **JOSBY_C_XX.** Don't press Enter yet, there is more to add.

Let's say your CEO wants positive confirmation over on your target machine that your HABP software has replayed all of the daily sales activity for each store.

__ C. Enter **'Minneapolis Store Sales Complete'** in the **Entry data** parameter. Include the single quotes. If you would like, you may enter a 'Journal entry type' parameter on the line above to tell you what kind of Entry you made. For example, if you would like to reiterate that this is a 'End of Day' (the Journal code 'U' designates it as a user Entry) you could enter a **Journal entry type** of **'ED.'**

Here comes the new option for V5R2:

__ D. Page down to the **Override journal state** parameter. Enter ***STANDBY.** Press **Enter.** We are telling the machine to override the normal Standby mode for this ONE special Journal Entry.

Let's see whether this critical application phase marker flowed out to our Journal despite the fact that it is in Standby mode.

__ 4. Type **DSPJRN JOSBY_C_XX/QSQJRN.**

Do you see the Entry you just added? You should see a Code of 'U,' (since it is a user provided rather than an OS/400 generated Journal Entry) and if you set the 'Journal entry type,' the code you chose should show up under 'Type.' Otherwise, under Type there will be '00.'

__ 5. Arrow down to your new Entry. Type **5**. Press **Enter**. You can see that it is a 'User generated entry,' along with the 'Entry specific data,' in this case 'Minneapolis Store Sales Complete.' Use the **F3** key to exit.

So let's reflect. What have we learned? Standby mode can really save you time and resources if you use a backup system for your data and need it as a production machine should your original production system go down. It's remarkably easy to turn on and off. You can still get your critical Entries deposited. And... your brother-in-law has lost another bet. You feel good, don't you?

Caching, Standby... they're <u>both</u> performance winners. Caching makes sense for Batch Jobs on your production system. Standby makes sense on your target (backup) system.

And how do you get your hands on these two performance gems? Simply install option 42 of OS/400 for V5R2. Guess you and your boss have a few things to talk about when you return home.

V5R2 Newest Journal and Commit Features

Lab 3: SMAPP Visibility and Tuning

IBM eserver iSeries

Lab 3. SMAPP Visibility and Tuning

This lab will provide an introduction to recent enhancements provided for SMAPP (System Managed Access Path Protection).

Introduction

SMAPP is a patented feature of OS/400 which allows the system to protect access paths within your Database so that these access paths do not need to be rebuilt after an abnormal end (i.e., a crash!). Rebuilding of such access paths can contribute a large amount of time to restarting the system. Most folks wouldn't want that experience. Thus, they're awfully glad that SMAPP exists. SMAPP implicitly journals only those access paths on your system which contribute the largest amount of rebuild time so that these selected indexes can be recovered from the Journal rapidly instead of being laboriously rebuilt from scratch. Each shop may specify their customized desired rebuild time to manage the balance between recovery time and the run time impact of protecting such access paths.

This Lab exercise will use the terms Access Path and Index interchangeably. SQL calls these objects Indexes. Native interfaces often refer to them as Access Paths.

Objectives

This lab teaches you how to:

- Make wise tuning choices based on the new screens provided in V5R2
- View and change the SMAPP threshold setting
- Use the new V5R2 "Protected" and "Not Eligible" SMAPP screens
- Use the new *INCHIDENT parameter on the DSPJRN command to view hidden Journal Entries

Lab Information

The notation XX that appears in library names, profile names, and so on, refers to your Team Number (for example, JOTEAMXX, JOLABXX, JOSMAPPXX). Refer to your lab worksheet for details.

Optional steps are included in this lab in addition to the required steps. These steps are not required to complete the lab, but can be attempted to further demonstrate the usage of various Journal commands. Optional steps are denoted with "Optional:" instead of the step number.

Lab Prerequisites

Before you begin this lab, be sure the following prerequisites are available:

- An IBM eServer iSeries or AS/400 with OS/400 V5R2, or higher, with:
  - 5722-QU1 -Query for AS/400
  - 5722-ST1 -DB2 Query Manager and SQL Development Kit for AS/400
- The JOLABXX library containing a data file for the lab.

Time required

The time required to efficiently complete this lab ranges from 30 to 40 minutes.

Task 1: Displaying the Default SMAPP Journal

This task will demonstrate how SMAPP behaves on the system.

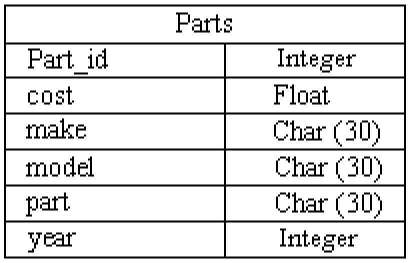
You own one of the largest auto salvage yards in the area (despite the fact that your seventh grade teacher said that anyone with that much "junk" in their locker wouldn't amount to much) and you have just finished getting every one of your parts entered into your new iSeries system. Let's take a look at what this data looks like....

__ 1. Start interactive SQL with the **STRSQL** CL command.

__ 2. Create our SQL collection with the **Create collection JOSMAPPXX** SQL statement. An SQL collection creates a native library, a Journal named QSQJRN, and various SQL related objects. Database objects created within the collection will automatically be journaled to the Journal QSQJRN. All of these objects take time to create, so wait just a moment as you gaze out your window at your vast array of junk (err... we mean parts).

__ 3. Exit interactive SQL with the **F3** option and then select option **1.**

__ 4. Let's move the 'Parts' file we've pre-populated for you from the JOLABXX library to your newly created SQL collection JOSMAPPXX. Type **MOVOBJ OBJ(JOLABXX/PARTS) OBJTYPE(*FILE) TOLIB(JOSMAPPXX).** The file you are moving is an SQL table which looks like the table shown below:



Parts	
Part_id	Integer
cost	Float
make	Char (30)
model	Char (30)
part	Char (30)
year	Integer

__ 5. Before entering Interactive SQL, let's note the current time according to the system clock. You will use this information in step 12. **DSPSYSVAL QTIME**.

Enter the current system time here: ____________________

__ 6. Start interactive SQL with the **STRSQL** CL command.

__ Optional: To view the initial contents of the database type **Select * from JOSMAPPXX/PARTS.** Exit the screen with F3 after display.

Reminder: You can use F20 (F8 + Shift) to scroll to the right.

This is a big sucker housing a whopping 50 parts (we had to keep this reasonable for Lab purposes - but your files are probably much larger!). With a big file like this that is accessed every time you buy or sell a part, you're going to want an index built over the file to help speedup performance. This will be helpful when you need to perform random lookups -- as you'd do if a customer called and asked if you had a 4-barrel carburetor for a '72 Cutlass Supreme.

__ 7. Let's create an index over the Parts file. **CREATE INDEX JOSMAPPXX/PARTIX ON JOSMAPPXX/PARTS(PART_ID, COST).**

The index is needed to improve the performance of the queries that are commonly run using this file.

Someone just brought in a muffler and tailpipe from a '79 Ford Pinto and you've decided to add them to your vast collection. Let's makes the updates to our file...

__ 8. Enter the following SQL statements to insert the records into the Parts file:

Insert into JOSMAPPXX/PARTS values(1001, 20.00, 'Ford', 'Pinto', 'Muffler', 1979)

Insert into JOSMAPPXX/PARTS values(1002, 10.00, 'Ford', 'Pinto', 'Tail pipe', 1979)

__ 9. Exit interactive SQL with the **F3** option and then select option **1.**

There sure is a lot of commotion around the junkyard. Butch, the junkyard dog, loves to chase gophers and keeps knocking the power cord out of the iSeries! With all of the activity around here and these large indexes which would need to be rebuilt, we should be very concerned with how long it will take our system to recover.

In order to quickly recover our data in the event of a system outage, your Lab instructors have preset the SMAPP time on our Lab system to *MIN. This will assure that in the event of a power outage your index will be recovered from the Journal instead of experiencing a long rebuild. That's the beauty of SMAPP.

__ 10. Type **DSPRCYAP** and press **Enter** to verify that the SMAPP "System access path recovery time" is still set to *MIN. Press **F3** to exit the DSPRCYAP screen. This SMAPP threshold can be altered with the EDTRCYAP CL command. (We obviously don't want you to tinker with this value now since we're all sharing the same system for this lab!)

The EDTRCYAP CL command is the method you use to control SMAPP activity on the system. You simply specify the amount of time you are willing to spend rebuilding access paths after a system crash. No additional setup steps are required! It's that simple. The system automatically will protect enough access paths to meet this threshold. You can specify a time in minutes, or the special value of *MIN as we have done here to protect all access paths on the system.

Note: *MIN may be a bit too aggressive for some of your shops. A more reasonable value of 30 minutes or even 40 minutes might be more appropriate. We selected *MIN for this Lab exercise so as to help put the "Pedal to the Metal" and deliberately make our background SMAPP tasks work hard.

__ 11. Look in your SQL-provided user Journal (that's the QSQJRN Journal residing in your collection) for evidence that the index is being protected. If the index were protected, we would see Journal Entries for both our file and index, wouldn't we? Use **DSPJRN JRN(JOSMAPPXX/QSQJRN)** to view the Journal Entries and see if both varieties exist.

Hmm.... no Entries for the objects here. How can that be? Well... that is because the underlying physical file (Parts) for our index is not being journaled since we just copied it into our collection (library).

An index must be journaled to the same Journal as the physical file it is built over. If SMAPP decides to journal an index such as ours which is not already explicitly journaled, it will instead clandestinely journal both the file and the index to the default hidden SMAPP Journal. That's the state our Table and Index are in right now.

In fact, such a system Default Journal exists on each ASP of the system. When your Parts file isn't being explicitly Journaled by you, this Default Journal will be used by SMAPP to protect your largest Indexes. I guess we need to look behind the curtain. Let's see what the system's Default Journal looks like. Will our Entries be in the Default SMAPP Journal? Let's go take a look.

__ 12. Are you ready to learn a deep dark secret that most folks outside of IBM's Lab don't even realize they can do? Good! The JRN parameter of the DSPJRN command has a special value *INTSYSJRN. Bet you've never used this option, huh? If you prompt the DSPJRN CL command, you'll see this choice listed. The problem is that using this option requires *ALLOBJ authority, and we weren't about to give you that authority on this lab system! Instead, we have a simple program that will invoke this DSPJRN command for you. Call this program using the **current date** and specifying the **time you recorded in step 7** like the following example:

CALL JOLABXX/JODSPJRN parm('02/11/2003' '11:30:00')

The main statement of the program that you are invoking looks like the following:

```
DSPJRN JRN(*INTSYSJRN) JRNID(10001) FROMTIME(&DATE &TIME)
```

Notice that we've specified a rather mysterious parameter JRNID(10001). Here's the deep dark secret. The first two characters of the code represent the journal type. The 10 tells the system you want to see the SMAPP Journal Entries, while the last three, 001, identify the specific ASP.

You can learn more about SMAPP and the *INTSYSJRN by reading more on this topic within the V5R2 Infocenter at:
http://publib.boulder.ibm.com/iseries/v5r2/ic2924/info/rzaki/rzakismappintro.htm

If you don't see any Journal Entries listed on your screen, double check that you've really entered Today's date and the time prior to the time you made your changes.

Can you find the 'PT' Entries from the last changes to your file? (It may be difficult to determine which entries are yours due to the fact that everyone will be sharing the same default Journal. It is even more difficult because the object name and other identifying information is not collected in the default Journal because these entries are only needed by the system for recovery) See! You've already learned something new in this lab exercise that most shops haven't even thought about! There is a hidden Journal on each ASP and you now know how to find it and view its contents. Imagine the interesting conversations you can now conduct around the water cooler or at your next Local User's Group meeting when you return home! Note: In this example, there is really no way for you to find your particular Entries, since this is a "short" Entry - i.e. No user, file, program, etc. information is included.

SMAPP is good, but it's only a safety net. SMAPP also takes some extra resources. An even wiser choice is to explicitly Journal your largest and most important tables as well as the access paths built over them. For your Junkyard that would be the 'Parts' file and 'PartIx' index.

Why don't we start <u>explicitly</u> journaling the Parts file to our SQL-provided Journal (QSQJRN), make some changes, and take a look at our user Journal again.

__ 13. Use the **STRJRNPF FILE(JOSMAPPXX/PARTS) JRN(JOSMAPPXX/QSQJRN)** CL command to start journaling 'Parts'.

This has the effect of ripping the Parts file away from the clutches of the hidden Default SMAPP Journal whose contents you viewed above and directing our Parts file's subsequent Journal Entries instead toward the more traditional SQL-provided Journal: QSQJRN.

__ Optional. Let's note the current time again according to the system clock. You will use this information in an upcoming optional step. **DSPSYSVAL QTIME**.

Enter the current system time here: ____________________

The phone is ringing, let's pick it up and see if we can make a sale.

__ 14. Start interactive SQL with the **STRSQL** CL command.

Someone has just ordered a muffler and tailpipe for their '79 Pinto! Wow, that was fast. We barely entered that part a few minutes ago. Let's hope we make a nice profit on this sale. Let's remove these items from our file.

__ 15. Enter the following two SQL statements to remove these records.

Delete from JOSMAPPXX/PARTS WHERE PART_ID = 1001

Delete from JOSMAPPXX/PARTS WHERE PART_ID = 1002

You should see a confirmation message that these rows have been deleted.

__ 16. Exit interactive SQL with the **F3** option and then select option **1**.

Now let's look again and see what our SQL-provided user Journal (QSQJRN) looks like. Will any of these new Entries be in the SQL Journal? Let's go see.

__ 17. Use **DSPJRN JRN(JOSMAPPXX/QSQJRN)** to view the Journal Entries and see what Entries exist. Unlike the SMAPP Journal, you will not see any access paths Journal Entries displayed here. Task 3 of this lab explains this phenomenon.

You should now be able to see some recently deposited Entries of type 'DL.' These Journal Entries show that a record was deleted at that point. You can display the specifics of the Entry by typing a 5 and pressing the Enter key next to the Journal Entry you would like to see. Obviously our SQL-provided user Journal (QSQJRN) is now picking up the changes you make to both your file and index instead of having them routed to the default SMAPP Journal. And why? Because the Parts file has been explicitly journaled to your user Journal.

While you have this screen up, take a look at the Journal Sequence numbers. Are there any skipped numbers? Do you know why? If not, stay tuned, we're going to let you in on that secret too!

The SMAPP activity is managed by SLIC tasks operating in the background. The background SLIC tasks which SMAPP employs to journal your largest indexes obviously consume some system resources, so you will want to carefully consider the balance between minimal recovery times and low run-time performance overhead if you want your system to maintain desired levels of performance. One way to balance IPL recovery time vs. performance is to <u>explicitly</u> journal any access paths you know you want protected, rather than allowing SMAPP to implicitly do so. Why is this better? Because SMAPP doesn't need to repeatedly, day after day, make the decision if you explicitly journal an index.

Hint: If you want short IPL / Recovery times for your most critical access paths (those most important to your business -- and your boss) explicitly Journal them. Doing so gives you faster IPL processing with less runtime overhead then expecting SMAPP to discover this need for you. How do you accomplish this? Answer: STRJRNAP.

__ Optional: Verify that the Entries are not also flowing into both your SQL Journal and the hidden default SMAPP Journal by running the program to display the default SMAPP journal again using: **CALL JOLABXX/JODSPJRN parm('*current date - MO/DA/YEAR* format' '*time before your deletes - HH:MM:SS format*')**. Remember that this Journal is used by everyone using this ASP, so you may see the Entries from others taking the Lab!

Task 2: New SMAPP 'Protected' Index Screen

We've discovered, above, that SMAPP tasks lurk in the background and periodically start and stop journaling protection for access paths of its own choosing in an effort to meet your specified target IPL/Recovery time. But which access paths is it protecting? We thought you would never ask....

This task will show off the new screen (yup! it's new for V5R2) which can be used to determine which access paths are currently protected by SMAPP. It provides a snapshot in time and thereby lets you see what these sly background SLIC tasks have been up to.

__ 1. Type **DSPRCYAP** and press **Enter** and Verify that the 'System access path recovery time' is still set to ***MIN**. Don't exit the DSPRCYAP screen yet.

__ 2. Open another session to the machine you are currently running on, using the same username as provided on your lab worksheet.

__ 3. *2nd window:* Start interactive SQL with the **STRSQL** CL command.

__ 4. *2nd window:* Insert a couple of records into our table to open it and keep it open using the SQL statements provided below. (DB2 on iSeries is extra smart about deciding when to leave files open. The file will be implicitly closed if only one SQL initiated change is entered - because Interactive SQL thinks you're probably done and not coming back hence it closes the file so as to perform timely housekeeping, but the file will be left open after the second change - because now you've got Interactive SQL thinking you just might be serious about coming back. Hence we want you to insert at least two new rows as shown below.)

We want the file kept open because that will "Expose" our index and leave it in an exposed state. (Exposed means that there's a risk that a machine crash might have to rebuild this Index from scratch). When an index (access path) admits it's exposed, SMAPP swings into action and tries to mitigate the IPL duration consequence of this exposure. That is precisely what happens for this Lab Exercise. By leaving the index "exposed", we're going to be able to illustrate some additional V5R2 features.

Insert into JOSMAPPXX/PARTS values(1003, 50.00, 'Pontiac', 'Grand Am', 'Engine', 1984)

Insert into JOSMAPPXX/PARTS values(1004, 10.00, 'Pontiac', 'Grand Am', 'Trunk lid', 1984)

Remember your first car?, the one you were so proud to drive, the one you washed every day?, the one you hated to give up? I think I see it over there in the corner of the Junkyard and no one has yet inventoried its parts... Take a few minutes, for old time's sake, and enter a few of it's parts into our database.

Both the Table and the corresponding Index are now open and waiting for more adds into the Parts table. If the projected Rebuild time of your exposed indexes are of long enough duration to be of concern - SMAPP notices the access paths that have been affected (and left open) by your Interactive SQL session, enables Journaling implicitly under the covers for these Access Paths and marks them as 'Protected.' Since we've got the SMAPP target recovery duration objective for this Lab set to *MIN, SMAPP is going to strive to protect every access path in sight. Normally, SMAPP would elect to protect only the biggest ones and leave the smaller ones unprotected. The 'Protected' state means that SMAPP has decided that they qualify for implicit journaling (i.e. are both exposed and big enough to care about). Hence they're not going to take long to recover if you should crash. A new feature in V5R2 allows you to view the identity of these protected access paths (indexes). Let us see which access paths are protected right now.

__ 5. *1st window:* Press **F14** (Shift + F2) for 'Display protected access paths' from the DSPRCYAP screen. Select *ALL for the ASP on the first 'Display Protected Access Paths' screen by pressing **Enter**.

The 'Display Protected Access Paths' screen will show all of the protected access paths (system wide!) on one screen. This screen will show up to 500 protected access paths, ordering them from the greatest 'Estimated Recovery Time' to the least. Can you find 'PartIx' from your library?

If you've reached this part of the lab exercise at about the same time that others sitting around you in this Lab are trying similar steps, you're going to see not merely your own Indexes on this screen but those from your colleagues as well. But that's good news. It helps illustrate that when you return home you can take a snapshot of all such SMAPP protected Access Paths system-wide.

When you close your files, your Indexes cease to be exposed and hence will disappear from this screen.

__ Optional: The rebuild time that you see for your Index here is indeed puny. If you would like to see this grow, exit SQL using an **F3** followed by option **1** in your *2nd window*. Run the MANYPARTS program to insert data into the PARTS file by using a call such as **call JOLABXX/MANYPARTS parm('PARTS' 'JOSMAPPXX' 'n')** where n is the number of rows to insert (maybe a couple of thousand parts). Press **F5** on your *1st window* while this is running to refresh your view of protected indexes.

Leave the second window open for use in the task 4, below.

You can see the estimated (rebuild from scratch) recovery time for each exposed Index. If Butch knocks out that power cord right now, this is how long it would take to rebuild this index if it were <u>not</u> protected by SMAPP. This is an easy way to determine which access paths on your system require the longest rebuild time.

OK - we realize that the value you see on the screen may not be very impressive or scary, since the size of the index and hence its estimated rebuild duration needed to remain limited for Lab purposes (you didn't want to spend your whole day in here, did you?). When you return home you'll probably want to display this screen and our hunch is that you'll see substantially larger values for your production indexes. And what action should you take? Well... why not relieve SMAPP of the burden of protecting the biggest critical access paths by explicitly Journaling them yourself via a STRJRNAP CL command? Remember, however, that the information on this screen is a changing snapshot. It only shows currently exposed indexes. Hence, you'll want to sample this screen during the busiest part of your day.

__ Optional: *1st window:* We've talked a lot about these background SMAPP tasks. Let's see if we can catch one of these tasks running with the WRKSYSACT command. Only one user can run this command at a time, so if you don't get in you may have to try again later. Enter the **WRKSYSACT** CL command.

Press **F15** (Shift + F3) to show only tasks. Use the **F10** button to refresh the screen and see if you can catch one of the SMAPP tasks at work (JO-TUNING-TASK or JO-EVALUATE-TASK).

You will only see these tasks running if there is sufficient activity on the system when you are running this command. To learn how to use various performance tools to track exactly what these tasks are up to - see the 5th Lab on Performance Counters and Tracepoints!

Task 3: *INCHIDENT - Display Hidden Journal Entries

This task demonstrates our new V5R2 DSPJRN option to display formerly hidden, internal Journal Entries.

Have you ever looked at the sequence numbers associated with your Journal Entries and noticed any skips? Do you wonder why? In this task, you will find the answer to that age old question. Let's first take a look at your Journal and see where some of these skipped sequence numbers are.

__ 1. Type **DSPJRN JOSMAPPXX/QSQJRN** on the CL command line.

Can you see some skips in the numbers? Are you missing Entries? Have they been dropped somewhere? What is going on? Have car parts been going out the door without the changes being journaled? Hmmm.... This is a matter for a super sleuth. But hold the phone. Before you dial 911 and ask for the Bunko squad, let's take a closer look.

Actually, nothing is missing. There are some Entries which are not displayed. Would you like to see what they are? Well, now -- with the new support in V5R2 -- you can.

The *INCHIDENT (include hidden Entries) option on the DSPJRN CL command is new in V5R2. This option allows you to see previously hidden Entries in your Journal. These hidden Entries are the 'skips' you may see in the sequence numbers of your Journal Entries.

__ 2. Type **DSPJRN JOSMAPPXX/QSQJRN INCHIDENT(*YES)** on the CL command line to display all of the Journal Entries including the hidden Entries.

Now you can see all of the Entries that were previously hidden. There are no more missing or skipped sequence numbers. You're feeling empowered, right? The Entries that formerly were hidden from your view have the value *OMITTED in the 'Job' column. Hidden Journal Entries are simply Entries from internal OS/400 and SLIC operations involved in the execution of our Database requests. This includes index-related Entries (a.k.a. SMAPP induced Entries), internal format Entries (those that help DSPJRN figure out how to map internal representations, like date fields, to external humanly readable representations), and Entries used to track the location and quantity of deleted records in a file.

They've formerly been hidden (and that's still the default on DSPJRN) because we didn't want to clutter your screen with purely Internal OS/400 induced Journal Entries which mere mortals need not manage. Only the SLIC code needs to "see" them because he's the only one who responds at IPL time to their presence. However, for V5R2, we've elected to let you show your Auditor that such Entries aren't really missing, just hidden to help reduce clutter. Got a pesky auditor back home? Now you can put their mind to rest.

Ahhh.... you're feeling better already.

Task 4: New 'Not Eligible' for SMAPP Protection Screen

This task will demonstrate the new V5R2 screen you can use to display access paths which are not eligible for SMAPP to protect.

I know... you've already been in this lab for quite awhile and you are probably getting a little tired - but the contents of this task could be the most important thing you learn in this lab if your shop has access paths which are not eligible for SMAPP. But how would you possibly know if you had any of these access paths? Go ahead, stand up, and stretch if you need to before we show you how to do this.

Consider the following scenario: You have determined that you are willing to spend no more than 50 minutes rebuilding indexes in the case of a system crash so you have set your SMAPP target recovery value to 50 using the EDTRCYAP CL command. You have a large number of small indexes which frequently change and a few large indexes which stay open for long periods of time. These large indexes would have a combined rebuild time of greater than 50 minutes if they were not protected by SMAPP. Obviously it's the Big ones you want SMAPP to select.

In this case the normal SMAPP response would be to protect these large indexes in order to meet your specified recovery time while the large number of small, frequently changing indexes would not need to be protected in order to achieve your target. However, there's a subtle caveat: if your large indexes are not eligible for SMAPP protection, your system will immediately be over the 50 minute recovery threshold and SMAPP will respond by working extra hard to protect all of your small indexes. Yikes! The time to manage, start journaling, and record the changes for all of these many small indexes will be much more time-consuming than journaling a few of your large indexes.

Hmmm... what's going on here? Why did our intuition mislead us? Let's see if the new V5R2 screen can unravel this mystery.

__ 1. Type **DSPRCYAP** and press **Enter.**

__ 2. Verify that the 'System access path recovery time' is still set to ***MIN.** Exit using **F3**.

One way for an access path to be ineligible for any SMAPP protection is if the access path is built over a physical file which was created with the attribute FRCACCPTH(*YES). This particular option is an ancient option which achieves the same recoverability as journaling with a much higher overhead. (If any of your files have inherited this option from the past, you should turn this attribute off ASAP and explicitly journal the file and access paths instead.) You'll thank us for this advice. We will use this variety of ineligibility in our example (there are other causes documented at the end of this Lab exercise).

__ 3. Let's set up the problem by assigning this ancient and offensive option to our index. Type **CHGLF FILE(JOSMAPPXX/PARTIX) FRCACCPTH(*YES).**

Note: This should convince SMAPP that this Access Path already has other means of IPL protection and hence it should skip this Access Path when looking for access paths in need of protection.

Let's pretend you haven't attended this Lab and hence still use the ancient (may it rest in peace) option:

__ 4. *2nd window:* Start interactive SQL with the **STRSQL** CL command if your session does not already have Interactive SQL started.

__ 5. *2nd window:* Insert a couple of Entries into our table to open it and keep it open with the following two SQL statements.

```
Insert into JOSMAPPXX/PARTS values(3005, 15.00, 'Ford',
        'F150', 'Door', 1978)
```

Insert into JOSMAPPXX/PARTS values(3006, 25.00, 'Ford', 'F150', 'Alternator', 1978)

The file is now open and waiting for more records to be added into Parts. SMAPP considers our exposed access path 'Not eligible' for implicitly journal protection. Why? Because it notices the FRCACCPTH setting, bummer!

A new feature in V5R2 allows you to view these ineligible access paths. Let us see which paths are 'not eligible' right now.

__ 6. *1st window:* Enter **DSPRCYAP** and press **Enter**.

__ 7. *1st window:* Press **F13** (Shift + F1) for 'Display not eligible access paths.' Choose ***ALL** for the ASP to use by pressing **Enter**.

On the 'Display Not Eligible Access Paths' screen, you can see all of the currently exposed Access Paths 'not eligible' for SMAPP protection on one screen. This screen will show up to 500 ineligible access paths, ordering them from the greatest 'Estimated Recovery Time' to the least. Why do you think we elected to sort such that the biggest guys are at the beginning of the list?

It's obvious that objects on the top of this list which deserve our immediate attention... And what should we do if we find such access paths? Answer: Try to modify as many of them as practical so they no longer show up on this screen. If they're using the ancient FRCACCPTH(*YES) option, use CHGPF or CHGLF to turn off this ancient and wasteful setting. Doing so will improve both IPL duration and reduce CPU and disk overhead. It's a win-win situation. But please remember: this screen is only a snapshot in time. Just because you find no culprits on this screen at 2 AM on Sunday doesn't mean you might not have some exposed and Ineligible Access Paths show up at 10 AM on Monday.

Your goal when you return home should be to assure that few if any culprits show up on this screen. Notice that the final column on this screen shows you why a particular Access Path has ceased to be eligible for SMAPP protection.

__ 8. *2nd window:* Exit interactive SQL with the **F3** option and then select option **1**.

We thought you'll want to know some other ways an access path may end up being classified as ineligible:

- an access path built over a physical file which in turn is journaled to a Journal which is currently in *STANDBY mode (like the stuff we did in Lab #2) would be considered ineligible

- multi-format access paths whose underlying physical files are journaled to at least two different Journals would be considered ineligible (the solution? Cease using different Journals!)
- an access path residing in the QTEMP library would be considered ineligible
- a temporary access path created by Query, SQL, or DFU for the duration of the query
- an access path constructed as an SQL encoded vector flavored index
- you explicitly journal an access path by using the STRJRNAP command
- an access path defined over a database table that has MAINT(*REBLD) specified for its access paths

Not all of these reasons will show up on the 'Not Eligible' screen. In order to understand why, you must realize that there are really 3 categories of Ineligible access paths:

1. Access paths which do not need to be rebuilt at IPL and therefore do not need to be protected by SMAPP (access paths in QTEMP, temporary access paths whose life time is only the duration of a query, MAINT(*REBLD) access paths, and explicitly journaled access paths). SMAPP doesn't think about these, nor does it need to.
2. Access paths which cannot be Journaled (encoded vector indexes). SMAPP simply ignores these.
3. Access paths which can't be journaled due to user action (standby journal used, multi-format access paths, and FRCACCPTH(*YES) access paths).

Only the access paths in category 3 are the ones that are shown on the screen. The access paths in category 1 will not contribute to IPL recovery time. Hence you need to take no action on their behalf. The access paths in category two cannot be journaled so there's not a darn thing you or SMAPP can do about them. Only these access paths in category 3 are the ones which you can do something about! Once you go back home and use the new 'Not Eligible Screen' you will be able to make the necessary changes to make these access paths eligible for SMAPP protection and improve your overall system performance.

One more note for those of you considering using Standby Mode.... It's true that Standby mode makes indexes ineligible for SMAPP. That's a very deliberate choice on our part. Thus, if you plan on doing other activity on the target system in an HA (High Availability) environment system, you will want to use the other new V5R2 SMAPP option available on both the EDTRCYAP and CHGRCYAP commands to not include ineligibles in the estimated SMAPP recovery time. This will allow you to use Standby and not force SMAPP to Journal all of the other indexes on your system due to the large exposure resulting from the ineligible indexes journaled to your Journal in Standby mode. In essence, you're letting SMAPP off the hook by advising it that you don't mind rebuilding the Indexes associated with files that are being replicated.

You can specify that choice when you get back home (don't do it now... You'll mess up other users of this Lab) with the EDTRCYAP CL command. This command brings up a screen very similar to the DSPRCYAP screen that you have seen on which you can modify the SMAPP setting.

By now you should have learned quite a bit about SMAPP.   Hopefully you will be able to take some of the suggestions made throughout this lab back home with you and employ some new choices on your own systems!

V5R2 Newest Journal and Commit Features

Lab 4: Savepoints

IBM *server iSeries*

Lab 4. Savepoints

In this section of the lab SQL savepoints will be explored.

Introduction

Savepoints are a new SQL feature in V5R2 which can be used with commitment control transactions to set milestones within a transaction. Their presence allows an application to easily and selectively backout portions of a transaction without abandoning the entire transaction. And why would a Journal Lab exercise include savepoints? Information stored within your Journal facilitates such savepoint operations.

Objectives

This lab will:

- Demonstrate the usage of Savepoints
- Show you how to view the Journal Entries produced from a savepoint scenario

Lab Information

The notation XX that appears in library names, profile names, and so on, refers to your Team Number (for example, JOTEAMXX, JOLABXX, JOSAVPTXX). Refer to your lab worksheet for details.

Optional steps are included in this lab in addition to the required steps. These steps are not required to complete the lab, but can be attempted to further demonstrate the usage of various Journal commands. Optional steps are denoted with "Optional:" instead of the step number.

Caution: If you quit this lab before completing it, please turn commitment control off for your job. Having commitment control on may affect other labs which you continue to work on. To turn commitment control off: start interactive SQL with the **STRSQL** CL command; press **F13**; choose **1**, 'Change session attributes,' on the 'SQL Session Services Screen;' on the 'Commitment control' line enter ***NONE**; press **Enter** twice. Exit interactive SQL with the **F3** option and then select option **1**. Then run the **ENDCMTCTL** CL command.

Lab Prerequisites

Before you begin this lab, be sure the following prerequisites are available:

- An IBM eServer iSeries or AS/400 with OS/400 V5R2, or higher, with:
  - 5722-QU1 -Query for AS/400
  - 5722-ST1 -DB2 Query Manager and SQL Development Kit for AS/400
- The JOLABXX library contains an SQL script for the lab.

Time required

The time required to efficiently complete this lab ranges from 20 to 30 minutes.

Task 1: Savepoints - Releases, Rollbacks, and Nested

In this section of the lab you will create a set of database tables representing a travel agency's database. You will then tell the travel agent your plans and they will enter them into the database.

Your vacation time is coming up quickly. This year you decided to spend a week in Florida! You need to make reservations. You will need to book a hotel room, plane tickets, and don't forget the rental car!

So here we are at the travel agency. Let's take a look at their computer system as they enter your information.

__ 1. Start interactive SQL with the **STRSQL** CL command.

__ 2. Turn commitment control on. Commitment control must be used here for savepoints to work. Press **F13**. Choose **1**, 'Change session attributes,' on the 'SQL Session Services Screen.' On the 'Commitment control' line enter ***CHG**. Press **Enter** twice.

Hereafter, any changes made to the Travel Agency's database as part of booking your reservations will be flagged as tentative until you elect to close out the reservation by committing the transaction.

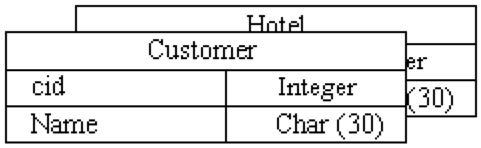
__ 3. Create an SQL collection with the **Create collection JOSAVPTXX** SQL statement. An SQL collection creates a native library, a Journal named QSQJRN, and various SQL related objects. Database objects created within the collection will automatically be journaled to the Journal QSQJRN. Be patient, this may take a while.

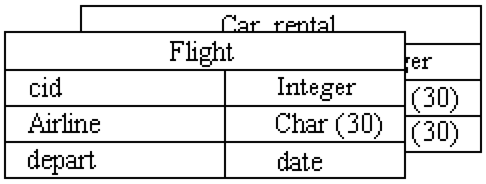
__ 4. Commit the creation of your new collection by entering **Commit.** This needs to be accomplished before we exit the SQL Interactive environment below. We need to assure that the creation of our collection ceases to be flagged as tentative before we execute RUNSQLSTM in step 6.

__ 5. Exit interactive SQL with the **F3** option and then select option **1**.

At this point we are going to create your travel agency's database tables.

__ 6. Create a set of database tables by executing the SQL statements residing in the SQLSPT member of the JOLABXX/SQLSTMT file. This can be accomplished with the **RUNSQLSTM SRCFILE(JOLABXX/SQLSTMT) SRCMBR(SQLSPT) DFTRDBCOL(JOSAVPTXX)** CL command. The script executed by this command contains the following SQL statements and will create the database tables shown below:





```
Create table Customer (cid int, Name char(30));
Create table Hotel(cid int, H_Name char(30));
Create table Flight(cid int, Airline char(30), depart date);
Create table Car_rental(cid int, Rental_Co char(30), car_type char(30));
```

__ 7. Start interactive SQL with the **STRSQL Commit(*CHG)** CL command. This informs the operating system that any records in the database that are added or changed (*CHG) are to remain locked up until we make our final decision to commit the transaction (in a sense it holds your tentative hotel room and airline seat for you). Commitment control must be enabled here since savepoints are a selective commit fallback mechanism.

Okay, the travel agent is ready to enter your information. First, your identity must be entered into the Customer database.

__ 8. Enter the **Insert into JOSAVPTXX/CUSTOMER values(1001, '*Your Name*')** SQL statement.

You have just given the travel agent your personal information. Should anything need to change later on, you don't want to have to answer all of those questions again, so we will enter a savepoint here. Now if anything needs to be changed later on, the agent can simply rollback to the point in time after your personal information was entered (rather than cancel the whole transaction).

__ 9. Set a savepoint, Cust_Data, by entering **SAVEPOINT Cust_Data unique on rollback retain cursors.** Now if you need to rollback changes after this point, you will only need to come back to here, rather than all the way back to the beginning of the transaction. By declaring the savepoint 'unique' you are specifying that the savepoint name cannot be reused within the unit of work.

'On rollback retain cursors' you are saying that you do not want cursors to be closed upon rollback to the savepoint if they are opened after the savepoint is set.

Alright, you are going to Florida. Do you know where you are going to stay? Well, let the travel agent know!

__ 10. Type **Insert into JOSAVPTXX/HOTEL values(1001, '*Your hotel name*')** on the Interactive SQL screen and execute this statement.

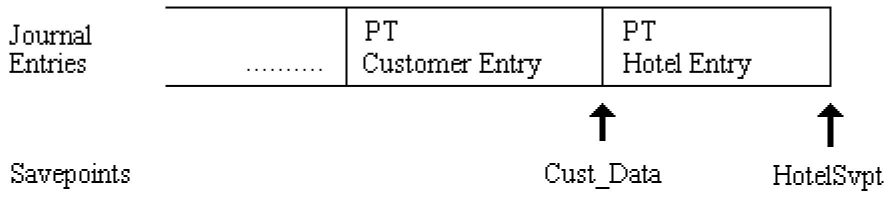
Hope you have picked a really nice hotel chain. No use skimping. You deserve it. It's by the beach, right?

Okay, there were rooms available, and that is settled. We haven't committed anything yet, but what if your flight is full or the rental car company who has sent you a discount coupon doesn't operate in Florida? We don't want to have to roll all the way back to the beginning of the transaction. Let's set up another savepoint, so if anything needs to be changed, hereafter we can just roll back to after the hotel Entry.

Savepoints can be stacked and nested, as shown here, so you can rollback to different stages of a transaction, depending on the changes you want to make.

__ 11. Set a savepoint, HotelSvpt, by entering **SAVEPOINT HotelSvpt unique on rollback retain cursors.**

Below is a timeline of what we have accomplished so far:



Now that your second savepoint is in place, why don't you choose a tentative car rental company and the car you want to drive. If you need to make changes to this later, you easily can do so using the rollback command.

__ 12. Type **Insert into JOSAVPTXX/CAR_RENTAL values(1001, '*Rental company*', '*car type*')** and kick off this statement**.** Go ahead. Pick a dream car. Something you've always wanted to drive. A convertible? A Corvette? A Viper? Name your personal favorite.

__ 13. Set another savepoint, RentalSvpt, by entering **Savepoint RentalSvpt unique on rollback retain cursors.**

Hereafter if there is anything that needs to be changed, you will still get your favorite car.

Okay, now you can book your flight. Pick an airline and a date, and let's see if there is anything available.

__ 14. Run the following SQL statement: **Insert into JOSAVPTXX/FLIGHT values(1001, '*Airline name*', '*departure date MO/DA/YR*').**

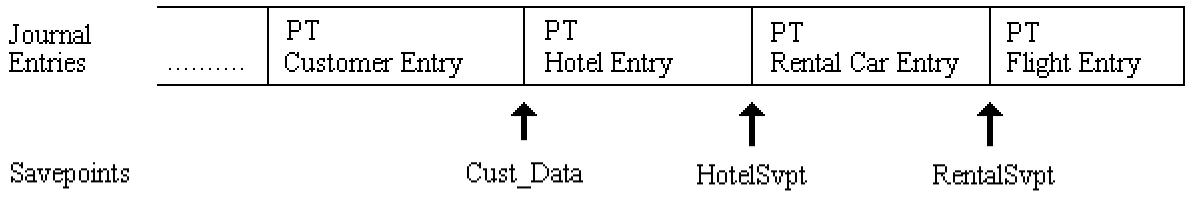
Oh no! There is a large convention in Florida the week you are going. This airline is full. In fact, all of the flights to this airport are booked. Luckily, there is another airport not far away. Unfortunately, your rental car company does not have an office and lot in that area. You will need to make a reservation with a different car company. But we sure don't want to perform a full rollback of the whole transaction, do we? No! That would put us back to the point we were before we even walked in the door. We'd have to start from scratch and re-register our personal information and lose our hotel room. We sure don't want that to happen! I guess it's a good thing we've been setting savepoints along the way.

As you can see from the diagram below, there are three alternative Savepoints we have established. We can selectively rollback to any one of them. In fact, your application actually has four Rollback choices at this point. Can you name the fourth?

Before we do a selective rollback to a particular savepoint, let's confirm that our changes for this tentative transaction have truly made it into the database.

__ 15. Enter **Select * from JOSAVPTXX/CUSTOMER, JOSAVPTXX/HOTEL, JOSAVPTXX/FLIGHT, JOSAVPTXX/CAR_RENTAL**. What do you see? Are all of your tentative choices visible in the Travel Agency's database? Remember, you can scroll to the right with F20 (Shift + F8).

Below is a timeline reminder of what we have accomplished so far:



You will want to cancel portions of your previous tentative reservations using the rollback command. The rollback command will rollback all changes occurring after the specified savepoint. Since you want to keep your hotel reservation, which savepoint will you want to rollback to? Yup! That's right... the second savepoint, HotelSvpt. You need to make new reservations only for your car rental and your flight.

When you rollback to savepoint, HotelSvpt, you are also discarding the savepoint, RentalSvpt.

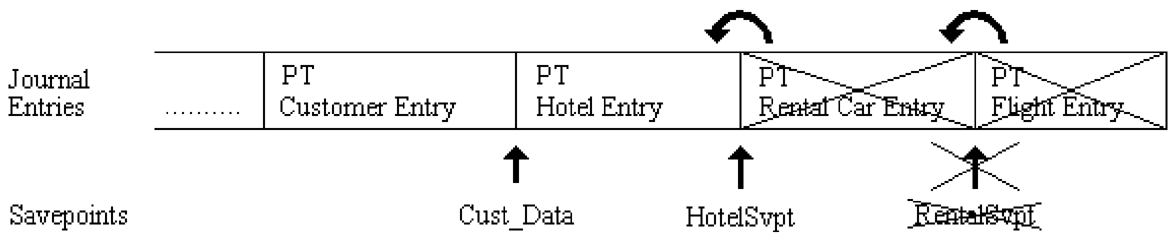
__ 16. I guess it's time we Rolledback the changes to both the Flight and Car_rental tables. Enter **Rollback to savepoint HotelSvpt**.

Let's see how this has changed our database:

__ 17. Run the following SQL select statement: **Select * from JOSAVPTXX/CUSTOMER, JOSAVPTXX/HOTEL**. You should see that your name, customer id, and hotel exist despite the rollback. Now type **Select * from JOSAVPTXX/FLIGHT, JOSAVPTXX/CAR_RENTAL**. What do you notice? The flight and car rental data has been removed from the tables, but (as you discovered in the previous select) the data entered before the Hotel savepoint is still there.

Gee! That's neat... got any application in your shop that could benefit from this kind of selective rollback?

Here's a pictorial representation of where we stand now:



Let's try picking another car rental company.

__ 18. Pick a rental car company by running the statement: **Insert into JOSAVPTXX/CAR_RENTAL values(1001, '*Alternative rental company*', '*car type*').**

Now you can book your flight at the alternate airport. Pick an airline and a date, and let's make sure there is something available.

__ 19. Insert a row into the "Flight" table by running **Insert into JOSAVPTXX/FLIGHT values(1001, '*Second airline name*', '*departure date MO/DA/YR*')**.

Everything is good this time. Your hotel, flight, and car rental are all set up. We no longer need the savepoints we set up. We could release them one at a time, starting with the most recent, or we could simply release the first savepoint, and all savepoints created thereafter would also be released.

Let's make life easier and simply release the first savepoint.

__ 20. Release the initial <u>savepoint</u>. Enter **Release savepoint Cust_Data**.

There's no selective going back now! You're Florida-bound!

We still need to commit the changes. A commit will automatically release all savepoints, so we did not necessarily need to release them when we did. However, were we to have had more transactions, i.e. billing work, to do afterward, we probably would have wanted to release these savepoints and create some new ones.

__ 21. Execute a **Commit** to commit the entire transaction.

That's it. Your reservation is in place. Bring along some sun screen and enjoy the trip! Look out for sharks, and... oh yeah, send us a postcard.

__ 22. Exit interactive SQL with the **F3** option and then select option **1**.

Task 2: Journal Entries Associated with Savepoints

In this task, we will take a look at the Journal Entries generated from the savepoint operations we performed earlier.

__ 1. Type **DSPJRN JOSAVPTXX/QSQJRN** and use the **Page Down** key to browse through the Entries you find there. Use **option 5** to explore the Entries that have been generated.

The SC and CM Entries are common Journal Entries during savepoint transactions, and tell you where the commits start and end, the PT and DR Journal Entries tell you where records were added, and where records were deleted during rollbacks.

The SC Entry is the start of your commit cycle, while the CM marks the commit, or end of your cycle. Following the PT (record added) for the hotel are two more PTs for the initial reservations you booked within the Car_rental and Flight tables. Next are two DR Entries for the records deleted when we rolled back to savepoint 2 (HotelSvpt).

If you are Journal savvy, you will probably recognize the PX Entries. This is the Journal Entry flavor used when inserted new records into deleted record locations in a file. If you would like more of an explanation - ask your lab instructors.

__ Optional: Why don’t you try making reservations for your vacation to Germany next summer? You can make your hotel, flight, and car rental reservations.

Don’t forget to add a savepoint after your hotel is booked. They fill up quickly, so you wouldn’t want to lose that great view of the Rhine if something needed to be changed later.

But wait! Your wife really wanted to experience the excellent public transportation system in Germany. She doesn’t want all the hassle of finding parking places and paying such high prices for gas. You had better rollback and get rid of that car rental. I know, I know... you had your heart set on seeing what “Pedal to the Medal” could do on the Autobahn where there are no speed limits. But trust us, the train ride along the Rhine is breathtaking and oh, so relaxing. Keep an eye peeled for the castles on the surrounding hills!

Don’t forget to rebook your flight, if you had booked it before you rolled back. Now commit your changes and take a look at your Journal Entries. Are they what you expected?

__ 2. Start interactive SQL with the **STRSQL** CL command.

It’s time for a bit of housekeeping:

__ 3. We need to turn commitment control off. Press **F13**. Choose **1**, ‘Change session attributes,’ on the ‘SQL Session Services Screen.’ On the ‘Commitment control’ line enter ***NONE**. Press **Enter** twice.

__ 4. Exit interactive SQL with the **F3** option and then select option **2**.

__ 5. Enter **ENDCMTCTL** on the CL command line.

7

V5R2 Newest Journal and Commit Features

Lab 5: Journal Performance Counters and Trace Points

IBM *e* *server iSeries*

IBM *e*server. For the next generation of e-business.

Lab 5. Journal Performance Counters and Tracepoints

In this section of the lab the newest Collection Services Journal performance counters and Performance Explorer (PEX) tracepoints will be explored.

Introduction

A number of brand new Journal Performance measurements are available in V5R2. These new measurements include new system wide Collection Services counters, new task-based (per job) Collection Services performance counters, and new Performance Explorer tracepoints. The addition of these counters are intended to assist in detailed performance analysis of scenarios involving journaling and SMAPP usage.

If you're serious about getting the best possible performance in your shop, you'll want to heed the advice given in the Redbook: "**Striving for Optimal Journal Performance**" found on the web at: *www.redbooks.ibm.com*, and you'll want to confirm the resulting performance benefits (or investigate lingering performance bottlenecks) by employing the new performance counters demonstrated here.

If you're a performance tool provider, you may want to fold some of these new counters into your product.

If you're a performance consultant, you may want to bone-up on these new performance counters so that you can tune the Journal environment more effectively for your customers.

Objectives

This lab teaches you how to:

- Enable and display the newest Journal Performance counters.
- Enable and display Journal PEX tracepoints

Lab Information

The notation XX that appears in library names, profile names, and so on, refers to your Team Number (for example, JOTEAMXX, JOLABXX, JOPFRXX). Refer to your lab worksheet for details.

Optional steps are included in this lab in addition to the required steps. These steps are not required to complete the lab, but can be attempted to further demonstrate the usage of various Journal commands. Optional steps are denoted with "Optional:" instead of the step number.

Reminder - To gracefully exit the lab:

If you quit this lab before completing it, please turn commitment control off for your job. Having commitment control on may affect other labs which you continue to work on. To turn commitment control off: start interactive SQL with the **STRSQL** CL command; press **F13**; choose **1**, 'Change session attributes,' on the 'SQL Session Services Screen;' on the 'Commitment control' line enter ***NONE**; press **Enter** twice. Exit interactive SQL with the **F3** option and then select option **2**. Then run the **ENDCMTCTL** CL command.

Lab Prerequisites

Before you begin this lab, be sure the following prerequisites are available:

- An IBM eServer iSeries or AS/400 with OS/400 V5R2, or higher, with:
  - 5722-QU1 -Query for AS/400
  - 5722-ST1 -DB2 Query Manager and SQL Development Kit for AS/400
- The JOLABXX library contains a program and an SQL script for the lab.

Time required

The time required to efficiently complete this lab ranges from 25 to 35 minutes.

Task 1: Journal Performance Counters

The set of tools known as Collection Services collects a broad range of system data, called Performance Counters, at regularly scheduled intervals, with minimal system resource consumption.

There are many new V5R2 Journal related counters, a complete list may be found at the end of this lab. A number of them will be demonstrated during this lab.

School will be starting soon and you have been asked to help design and tune a database for a new school supply store in town being opened by your mother-in-law. You want to make a good impression.

__ 1. Start interactive SQL with the **STRSQL** CL command.

__ 2. Create a collection, JOPFRXX, which will be used in this Lab by executing the **Create collection JOPFRXX** SQL statement. An SQL collection creates a native library, a Journal named QSQJRN, and various SQL related objects. Database objects created within the collection will automatically be journaled to the Journal QSQJRN. Be patient, this may take a while.

__ 3. Exit interactive SQL with the **F3** option and then select option **1**.

__ 4. Create two libraries which will be used to store your Performance results with the **CRTLIB JOBEFOREXX** and **CRTLIB JOAFTERXX** CL commands.

__ 5. Note the current time according to the system clock. You will use this information later. **DSPSYSVAL QTIME**.

Enter the current system time here: ____________________

__ 6. Type **WRKJOB** to get the name of your job in the upper left-hand corner of the 'Work with Job' screen. We're going to need that later, too.

Enter your job name here: _____________________

Collection Services is either on or off for an entire system. Your lab instructors enabled Collection services on this system for you using the 'go perform' menu.

__ 7. Create performance database files to view what your Performance Counters read prior to any actions by you. **CRTPFRDTA FROMMGTCOL(*ACTIVE) TOLIB(JOBEFOREXX) CGY(*JOBMI)**

FROMTIME(‘*today’s date - MO/DA/YR*’ ‘*current time from step 4 - HH:MM:SS*’). This will create and prime a set of database files in your library containing the thread level performance counters starting with the time you have specified. The CGY parameter of the command specifies the categories in the management collection object which will be processed into database files. Declaring *JOBMI as the category name thus specifies that Jobs (MI tasks and threads) will be processed into database files.

__ 8. Start interactive SQL with the **STRSQL** CL command.

__ 9. Use the following statement to view the current values of some Performance Counters (primary commit operations, user Journal SMAPP deposits, and primary decommit operations). **Select JBNAME, sum(JBCOP) Commits, sum(JBUJD) SMAPP_Deposits, sum(JBDOP) Rollbacks from JOBEFOREXX/qapmjobmi WHERE JBNAME = ‘*job name*’ group by JBNAME**. The job name must be in UPPERCASE.

The values of the counters are all zero! Right? Well of course they are...we haven’t done any work yet. Let’s make some changes which will bump these performance counters and then come back and take another look.

__ 10. Turn commitment control on. Commitment control must be used here for our subsequent commits to work. Press **F13**. Choose **1**, ‘Change session attributes,’ on the ‘SQL Session Services Screen.’ On the ‘Commitment control’ line enter ***CHG**. Press **Enter** twice.

Our first few supplies have arrived, the Fed Ex driver is unloading now.

__ 11. Create a table, perform some inserts, a rollback, and finally commit the changes. This will increment the counters related to these actions.

Create table JOPFRXX/Supplies (item char(30), price float)

Insert into JOPFRXX/Supplies values(‘scissors’, 3.00)

Insert into JOPFRXX/Supplies values(‘backpack’, 20.00)

Break time! Your mother-in-law has just walked in with a tray of cookies. Well, we haven’t gotten very far, but you would rather not have to start over if something went wrong while you were gone. Let’s commit it to make sure our data is safe.

Commit

Break's over! Back to work! (Gosh, your mother-in-law is a real stickler, isn't she! No taking a few extra minutes at break time with her!)
Look, the 'Grand Opening' sales flyer! Backpacks are only going to be $17.00! Well, our original price of $20.00 is already committed, we'll just have to update the Entry. Let's go ahead and do so. Then commit it again so we don't have to worry about it later.

Update JOPFRXX/Supplies set price = 17.00 where item = 'backpack'

Commit

Insert into JOPFRXX/Supplies values('nifty calculator', 5.00)

Your boss just came by to let you know that you won't be getting those nifty see-through calculators you just entered after all. You had better remove them from the inventory!

Rollback

...by the way, how much of what you've entered this morning will actually be rolled back at this point?

__ 12. Exit interactive SQL with the **F3** option and then select option **1**.

OK, let's reflect: we created a table, added some product, committed our changes, updated an Entry, committed again, and rolled one Entry back... Hmm... I wonder what the Journal Performance counters have to say about our work up to this point. Let's find out...

At this point, make sure that it has been at least 1 minute since you made your last set of changes in step 11. For the lab, we have set the Collection Services interval to 1 minute. In order to make sure that your changes have been recorded, you must wait at least this much time.

__ 13. Create performance database files to view what your Performance Counters read after these actions. Again, use the system clock time you recorded in step 5 and the "After" library, instead of the "Before" library as in step 7.
CRTPFRDTA FROMMGTCOL(*ACTIVE) TOLIB(JOAFTERXX) CGY(*JOBMI) FROMTIME('*today's date - MO/DA/YR*' '*current time from step 4 - HH:MM:SS*').

__ 14. Start interactive SQL with the **STRSQL** CL command

__ 15. Let’s view the current values of some Performance Counters (primary commit operations, user Journal SMAPP deposits, and primary decommit operations). Caution: When entering the ‘job name’ the query is case sensitive. Make sure you are entering your job name with the correct capital letters. **Select JBNAME, sum(JBCOP) Commits, sum(JBUJD) SMAPP_Deposits, sum(JBDOP) Rollbacks from JOAFTERXX/qapmjobmi WHERE JBNAME = ‘*job name*’ group by JBNAME.** Job name must be in UPPERCASE.

What do you see? Can you explain each count? Do they make sense? If you have an application back home, maybe one you didn’t personally write, and you’ve wondered how many Journal operations it performs or how frequently it decommits transactions, you now have an easy way to collect such statistics.

And... If you’ve got a mother-in-law who thinks you’re a slacker, you’ve now got the assistance of the Journal facility to help prove otherwise!

__ Optional: To see the full set of performance data that has been collected for each interval in this file for your job, execute the **Select * from JOAFTERXX/qapmjobmi WHERE JBNAME = ‘*job name’*** SQL Statement. Use F20 (Shift + F8) to scroll to the right to view all of the data and column headings.

__ 16. Exit interactive SQL with the **F3** option and then select option **1**.

There are a wide variety of Journal related performance counters available on both a system-wide and thread (TDE) based level. A full list of the <u>new</u> Journal related performance counters for V5R2 is included below:

Column	Description
System wide counters	
SYJOER	SMAPP evaluations requested
SYJOES	SMAPP evaluations serviced
SYJOIB	SMAPP index build time estimations
SYJOS1	Most popular Journal Entry type flushing Journal buffer
SYJOC1	Number of bundles terminated by first Entry type
SYJOS2	Second most popular Journal Entry type flushing Journal buffer
SYJOC2	Number of bundles terminated by second Entry type
SYJOS3	Third most popular Journal Entry type flushing Journal buffer
SYJOC3	Number of bundles terminated by third Entry type
Thread specific counters	
JBCOP	Primary Commit operations
JBCOS	Secondary Commit operations
JBDOP	Primary Decommit operations
JBDOS	Secondary Decommit operations
JBPJE	Physical Journal writes
JBNSJE	Non-SMAPP Journal Entries
JBUJD	SMAPP deposits to user Journals
JBSJD	SMAPP deposits to System (default) Journals
JBBFW	Bytes written to fixed (ordinary) Journal Receiver area
JBBFA	Bytes deposited to fixed (ordinary) Journal Receiver area
JBBTW	Bytes written to transient (hidden due to *RMVINTENT) area
JBBTA	Bytes deposited to transient (hidden due to *RMVINTENT) area
JBTWT	Cumulative Journal bundle wait time
JBTNW	Duration of Journal bundle waits

Task 2: Journal Tracepoints

The counters we've been examining give a high level view of what's going on and are less intrusive than tracepoints. Hence, they're often the first tool you'll want to employ if you're trying to gauge the performance characteristics of a Journal-rich environment. Once you know a performance problem exists and you decide to drill down and investigate the particulars, you're going to want to shift gears from counters and turn to tracepoints. Tracepoints help isolate and identify complex performance problems. In this section of the lab, you will be looking at an example of some of the new Journal tracepoints in action.

__ 1. Add a personal PEX definition. **ADDPEXDFN DFN(JOPEXDFNXX) TYPE(*TRACE) JOB(*) TRCTYPE(*SLTEVT) SLTEVT(*YES) JRNEVT(*ALL).** This definition will collect all of the Journal events for your job (thread) because you specified *ALL for the JRNEVT parameter and * for the JOB parameter.

Unlike Collection Services which must be enabled system-wide (and hence you didn't have a private copy), any number of PEX traces can be running on a system at any particular time. Therefore, each lab user can start and end their own PEX session.

__ 2. Start PEX. **STRPEX SSNID(JOPEXDFNXX) OPTION(*NEW) DFN(JOPEXDFNXX).**

__ 3. Start interactive SQL with the **STRSQL** CL command.

__ 4. Turn commitment control on if it is not already on. Commitment control must be enabled here so that our commits will be honored. Press **F13**. Choose **1**, 'Change session attributes,' on the 'SQL Session Services Screen.' On the 'Commitment control' line enter ***CHG** if it is not already set to this value from a previous lab. Press **Enter** twice.

A big semi is backing up to the loading dock and here comes your mother-in-law with a list of the new items coming in! Let's add some more of these school supplies to our pricing database.

__ 5. Continue inserting values into the database.

Insert into JOPFRXX/Supplies values('pencils', 1.50)

Insert into JOPFRXX/Supplies values('lunchbox', 7.00)

Lunch time! It's that special goulash you don't have the courage to tell your mother-in-law you don't really like! Hmmm.... the system sure seems slow today! Let's commit our Entries to make sure they are safe.

Commit

Lunch's over! Back to work!

Insert into JOPFRXX/Supplies values('book covers', 25.00)

Oops! You just entered the price incorrectly. The book covers should only cost $2.50! You had better fix them quickly! You do not want to incur the wrath of your mother-in-law.

Rollback

Insert into JOPFRXX/Supplies values('book covers', 2.50)

Commit

__ 6. Exit interactive SQL with the **F3** option and then select option **1**.

Whew! That's enough for one day. But before we leave, let's find out what the trace facility has to say about the Journal-related actions that have been occurring in the supplies Database.

__ 7. End PEX. **ENDPEX SSNID(JOPEXDFNXX) DTALIB(JOAFTERXX).** This may take just a little while to complete while it is putting all of the performance data into Database files.

The above command will end your PEX session and also create a set of database files in your library. One of these files, QAYUSRDFN, contains the Journal PEX tracepoint data. This data is unformatted within this file. To format this data, you need to invoke a neat little program we wrote just for this lab. You can find a copy of this Trace Point formatter in the Appendix.

__ 8. Call the PARSEJOPEX program which will parse the trace data into an easily readable representation. **CALL JOLABXX/PARSEJOPEX parm('JOAFTERXX').**

Let's take a look at the resulting data...

__ 9. Start interactive SQL with the **STRSQL** CL command

__ 10. To view the current tracepoint data: **Select record, type, subtype, cycleid, mjocommitops, mjodecommops from JOAFTERXX/JOPEXOUT**.

The tracepoints provide you with a variety of information from the time when the tracepoint was encountered. You could use this data for a variety of purposes depending on what you are trying to investigate. For example, you could determine how long Rollbacks are taking in your application using the Start and End Rollback Operations and the timestamps which come with these tracepoints. The CycleID can also be used to associate the Journal Entries involved in the same commit cycle.

Below is a table you can employ to decipher the Journal actions corresponding to the type and subtype you will find listed from the above select statement:

Type	Subtype	Description
20	2	Start of Commit Operation
20	3	End of Commit Operation
20	4	Start of Rollback Operation
20	5	End of Rollback Operation

20	6	Start of Commit Cycle (the SC Entry)
20	7	Start of Journal Background Task Object Force
20	8	End of Journal Background Task Object Force
20	9	Start of SMAPP Evaluation
20	10	End of SMAPP Evaluation

Some of the events tracked here are revealing the actions of background Housekeeping Journal tasks which operate at the SLIC level of OS/400 (check out the optional Task 3 to see these tracepoints at work!). For example:

◆ The events with subtype 7 and 8 refer to the background forces done by the JORECRA tasks. The JORECRA tasks on a system occasionally sweep through main memory to flush changed pages of journaled objects from main memory onto disk. By flushing out the changed pages to disk, fewer Journal Entries will need to be applied to your objects at IPL time. These JORECRA tasks may have an impact on the performance of your system. Section 4.2.2. of the Redbook "Striving for Optimal Journal Performance" shows how to adjust the JORECRA behavior.

◆ The events with subtypes 9 and 10 refer to some of the background activity by SMAPP (System Managed Access Path Protection). These trace points track each time an index is evaluated by SMAPP to determine if the index should be implicitly journaled. See Lab 3 for more details on SMAPP. Section 4.1 of the Redbook "Striving for Optimal Journal Performance" is a good source for information on SMAPP performance information.

__ Optional: With each tracepoint, additional data is also collected. View all of the data for a particular tracepoint with the following SQL statement: **Select * from JOAFTERXX/JOPEXOUT**.

In addition to this data, more generic data, such as the job name and timestamps, is available in other PEX results files.

__ 11. Turn commitment control off. Commitment control must be turned off for some of the other sections of this lab to work. Press **F13**. Choose **1**, 'Change session attributes,' on the 'SQL Session Services Screen.' On the 'Commitment control' line enter ***NONE**. Press **Enter** twice.

__ 12. Exit interactive SQL with the **F3** option and then select option **2**.

__ 13. Enter **ENDCMTCTL** on the CL command line.

Optional Task 3: Additional Journal Tracepoints

The previous task introduced you to some of the new PEX tracepoints available in V5R2. This task also mentioned some of the tracepoints that are now available to help you understand what the background Journal tasks are doing on the system. This Lab will allow you to capture some of tracepoints being produced by these background tasks and examine what is happening.

__ 1. Add a personal PEX definition. **ADDPEXDFN DFN(JOPEX2_XX) TYPE(*TRACE) JOB(*) TASK(*ALL) TRCTYPE(*SLTEVT) SLTEVT(*YES) JRNEVT(*ALL).** This time we have specified to collect tracepoints which are being produced by all tasks on the system in addition to our own job with the TASK(*ALL) option.

__ 2. Create a library which will contain your PEX results with the **CRTLIB JOPEX2_XX** CL command.

__ 3. Start PEX. **STRPEX SSNID(JOPEX2_XX) OPTION(*NEW) DFN(JOPEX2_XX).**

__ 4. At this point, we urge you to run any activity on the system which may drive the JORECRA tasks or the SMAPP evaluation tasks (perhaps you could repeat a section of your favorite previous lab). The RECRA tasks will force objects when a large number of changes have been made and the SMAPP tasks will evaluate indexes which are changing on the system. If others in the lab are running other activities on the system, they may cause you to collect data for these tasks since these are shared background tasks.

If you can’t think of anything to run, follow these steps (using a program from Lab 3) to produce some RECRA and SMAPP activity.

__ A. Start interactive SQL with the **STRSQL** CL command.

__ B. Create a table with the **CREATE TABLE JOPEX2_XX/PARTS (ID int, COST float, MAKE char(30), MODEL char(30), PART CHAR(30), YEAR int**) SQL statement.

__ C. Let’s create an index over the Parts file. **CREATE INDEX JOPEX2_XX/PARTIX ON JOPEX2_XX/PARTS(ID).**

__ D. Exit interactive SQL with the **F3** option and then select option **1**.

__ E. Create a Journal Receiver with the **CRTJRNRCV JOPEX2_XX/R1** CL command.

__ F. Create a Journal with the **CRTJRN JOPEX2_XX/J1 JOPEX2_XX/R1** CL command.

__ F. Start journaling for your Parts file using the **STRJRNPF JOPEX2_XX/PARTS JOPEX2_XX/J1** CL command.

__ G. Run the MANYPARTS program to create some journal activity. **Call JOLABXX/MANYPARTS parm('PARTS' 'JOPEX2_XX' '50000')**.

__ 5. End PEX by running the following command **ENDPEX SSNID(*SELECT) DTALIB(JOPEX2_XX)**. This will allow you to see if you have created any PEX events for your trace. If the event count column shows that events have been created for your trace, choose option **1** to end your trace. If you haven't created any events yet, go back to step 4 and try again.

__ 6. Call the PARSEJOPEX program which will parse the trace data into an easily readable representation. **CALL JOLABXX/PARSEJOPEX parm('JOPEX2_XX').**

Take a look at the data in the output file using the techniques learned in the previous lab (Hint: you '*' in your SQL statement to see all of the data in the JOPEXOUT file or look at the program source in the Appendix). Can you determine if any SMAPP or RECRA background task behavior took place during your trace? Refer back to the table in Task 2 to see the definitions of the various tracepoint subtypes. Feel free to ask your lab assistants if you would like any help.

By examining data such as this, you can determine a variety of information such as the number of SMAPP evaluations that are occurring, the number of RECRA forces, the amount of time spent doing the activities, and the objects involved in these activities. These may be helpful in determining the exact cause of a performance slowdown on your system.

Appendix A

This Appendix contains various sample programs and SQL scripts which have been used throughout the labs. These example programs have not been subjected to any formal testing. They are provided "AS-IS" and they should be used for reference only.

UPDVAC Source Code

```
/*
 *  This C program will run database operations for the
 *    specified file of change requests.  This program contains
 *    statements which are valid only for the
 *    database environment specific to the Journal Lab.
 *
 *  Syntax:
 *     UPDVAC file
 *          file - the name of a Requests file
 *
 *  Compile Statement:
 *      CRTSQLCI OBJ(JOLABXX/UPDVAC)
 *               SRCFILE(JOLABXX/UPDVAC)
 *               COMMIT(*NONE) OBJTYPE(*PGM)
 *
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

EXEC sql include SQLCA;
EXEC sql include SQLDA;

int main(int argc, char *argv[])
{

  EXEC SQL BEGIN DECLARE SECTION;

  short int req = 0;
  short int counter = 1;
  short int empid = 0;
  char updstmt[1000];
  char selstmt[1000];
  char sel2stmt[1000];
  char file[30];

  EXEC SQL
    DECLARE c1 CURSOR FOR exestmt;
  EXEC SQL
    DECLARE c2 CURSOR FOR actstmt;

  EXEC SQL END DECLARE SECTION;

  /******************************************
   *   Parsing command line / error checking
   ******************************************/

  if (argc != 2)
  {
    printf("UPDVAC Requests file");
  }
  else
  {
    sprintf(file, "%s", argv[1]);

    sprintf(selstmt, "SELECT COUNT(*) FROM %s", file);
    EXEC SQL
      PREPARE actstmt FROM :selstmt;
    EXEC SQL
      OPEN c2;
    EXEC SQL
      FETCH c2 INTO :req;
    EXEC SQL
    EXEC SQL
      CLOSE c2;

    sprintf(sel2stmt, "SELECT emp_id FROM %s", file);

    EXEC SQL
      PREPARE exestmt FROM :sel2stmt;
    EXEC SQL
      OPEN c1;

    while(counter <= req)
    {

      EXEC SQL

        FETCH c1 INTO :empid;



      sprintf(updstmt,

        "UPDATE Vacation SET vac_days = vac_days - 1 WHERE emp_id = ?");


      EXEC SQL

        PREPARE newstmt FROM :updstmt;

      EXEC SQL
        EXECUTE newstmt USING :empid;

      counter = counter + 1;
    }
    EXEC SQL
      CLOSE c1;
  }

  return 0;
}
```

SQLSTMT/SQLDSP SQL Script

```
Create table Monday (emp_id int, reqdata date);
Create table Tuesday (emp_id int, reqdata date);
Create table Wednesday (emp_id int, reqdata date);
Create table Vacation (emp_id int, Name char(30), vac_days int);
Insert into Monday values (1001, '04/05/02');
Insert into Monday values (1002, '05/15/02');
Insert into Monday values (1003, '07/03/02');
Insert into Monday values (1004, '02/25/02');
Insert into Tuesday values (1005, '10/28/02');
Insert into Tuesday values (1006, '08/02/02');
Insert into Tuesday values (1007, '06/21/02');
Insert into Tuesday values (1008, '06/21/02');
Insert into Wednesday values (1001, '04/06/02');
Insert into Wednesday values (1005, '02/24/02');
Insert into Wednesday values (1005, '02/25/02');
Insert into Wednesday values (1004, '02/24/02');
Insert into Wednesday values (1005, '07/01/02');
Insert into Wednesday values (1005, '07/02/02');
Insert into Wednesday values (1005, '10/26/02');
Insert into Wednesday values (1005, '11/12/02');
Insert into Wednesday values (1005, '09/08/02');
Insert into Wednesday values (1005, '12/12/02');
Insert into Wednesday values (1005, '08/30/02');
Insert into Wednesday values (1005, '03/15/02');
Insert into Wednesday values (1005, '04/21/02');
Insert into Wednesday values (1005, '04/22/02');
Insert into Vacation values(1001, 'Bill', 5);
Insert into Vacation values(1002, 'Sue', 7);
Insert into Vacation values(1003, 'Sam', 4);
Insert into Vacation values(1004, 'Beth', 8);
Insert into Vacation values(1005, 'Cody', 30);
Insert into Vacation values(1006, 'Michelle', 6);
Insert into Vacation values(1007, 'Tom', 3);
Insert into Vacation values(1008, 'Jane', 9);
```

BUNDLE Source Code

```
/*
*  This C program will read an DSPJRN outfile of *TYPE5 (new for V5R2) and
*     return bundling information.  You will receive an error from this
*     program stating that the JOARM column is missing if you fail to
produce
*     the OUTFILE using the *TYPE5 option.  Use the DSPJRN INCHIDENT(*YES)
*     option for accurate bundling information.  This will allow
*     for the hidden entries to be included in your outfile which
*     also take up space on the Journal bundles.
*
*  The following is an example of a DSPJRN command with the options
*     required:
*       DSPJRN JRN(LIB/JRN) OUTPUT(*OUTFILE) OUTFILFMT(*TYPE5)
*            OUTFILE(LIB/OUTFILE) INCHIDENT(*YES)
*
*  Syntax:
*     BUNDLE lib file
*          lib - library
*          outfile - dspjrn outfile
*
*  Compile Statement:
       CRTSQLCI OBJ(JOLABXX/BUNDLE)
                SRCFILE(JOLABXX/BUNDLE)
                COMMIT(*NONE) OBJTYPE(*PGM)
*
*/
```

```
/* include the necessary C header files */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>     /* toupper() call */

/* Statements required for embedded SQL */
EXEC sql include SQLCA;
EXEC sql include SQLDA;


/* Begin main function */
int main(int argc, char *argv[])
{

 EXEC SQL BEGIN DECLARE SECTION;


 int i;   /* a simple counter */

 long int numents = 0;  /* number of entries in outfile */
 long int curr_ent = 1; /* current entry being processed */
 short int armnum = 0;  /* arm number of current entry */
 long int entsize = 0;  /* size of current entry */

 short int formerarmnumber = -1;   /* arm number of previous entry */
 long int bundlesize = -1;       /* running total size of current bundle */
 long runningnumbundles = 0;     /* current number of bundles found */
 long int totalsize = 0;         /* cumulative size of all entries */
 long int maxbundle = -1;        /* current maximum bundle size */
 long int minbundle = 9999999;   /* current minimum bundle size */

 char selstmt[200];     /* character arrays to hold our select statements
*/
 char sel2stmt[200];
 char lib[11];          /* character array to hold input library name */
 char outfile[11];      /* character array to hold outfile name */

 EXEC SQL
   DECLARE c1 CURSOR FOR exestmt;   /* declare SQL cursor C1 */
 EXEC SQL
   DECLARE c2 CURSOR FOR actstmt;   /* declare SQL cursor C2 */

 EXEC SQL END DECLARE SECTION;

 /******************************************
  *   Parsing command line / error checking
  ******************************************/

 if (argc != 3)
 {
   printf("ERROR - proper syntax is: BUNDLE lib outfile");
 }
 else
 {

   /* extract the library from the first argument */
   sprintf(lib, "%s", argv[1]);
   /* extract the outfile from the second argument */
   sprintf(outfile, "%s", argv[2]);

   /* convert the library to upper case */
   for (i = 0; i < 10; i++)
   {
      lib[i] = toupper(lib[i]);
   }
   /* convert the outfile to upper case */
   for (i = 0; i < 10; i++)
   {
      outfile[i] = toupper(outfile[i]);
   }

   /* create the SQL statement to determine the number of entries */
   sprintf(selstmt, "SELECT COUNT(*) FROM %s/%s", lib, outfile);
   EXEC SQL
     PREPARE actstmt FROM :selstmt;
   EXEC SQL
     OPEN c2;                      /* open the SQL view (cursor) */
   EXEC SQL
     FETCH c2 INTO :numents;       /* set the number of entries */
   EXEC SQL
     CLOSE c2;                     /* close the SQL cursor */

   /* output the number of Entries to the screen */
   printf("\n\n\n\n\n\n\n\n");
   printf("number of entries = %d\n", numents);

   /* Create SQL query which will pull two columns from your OUTFILE:
    *   the arm number on which each Journal entry resides and the
    *   width in bytes of each Journal entry.   These will be sorted
    *   by the Journal Sequence number so that we see consecutive Journal
    *   Entries in the order in which they were deposited.
    */
   sprintf(sel2stmt,
           "SELECT JOARM, JOENTL FROM %s/%s order by JOSEQN",
           lib, outfile);
   EXEC SQL
     PREPARE exestmt FROM :sel2stmt;
   EXEC SQL
     OPEN c1;                      /* open the SQL view (cursor) */

   /* loop through each entry in the outfile */
   while(curr_ent <= numents)
   {
     /* get the next available entry */
     EXEC SQL
       FETCH c1 INTO :armnum, :entsize;    /* extract the arm number width
*/

     totalsize = totalsize + entsize;

     /* test if current entry is start of new bundle */
     if (armnum != formerarmnumber)  /* provided that we have seen a bundle
*/
     {
       /* maintain overall stats if we really just finished a bundle */
       if (formerarmnumber != -1)
       {
         if (bundlesize > maxbundle)
            maxbundle = bundlesize;
```

```
          if (bundlesize < minbundle && bundlesize != -1)
             minbundle = bundlesize;
        }

        /* maintain stats considering this new bundle */
        runningnumbundles = runningnumbundles + 1;
        bundlesize = entsize;        /* prime the size of the new bundle */
      }
      /* else this entry belongs to the current bundle */
      else
      {
         bundlesize = bundlesize + entsize;
      }

      /* track the arm of the current entry */
      formerarmnumber = armnum;
      curr_ent = curr_ent + 1;
    }
    EXEC SQL
      CLOSE c1;

    /* make sure to add our last bundle into the statistics */
    if (bundlesize > maxbundle)
       maxbundle = bundlesize;
    if (bundlesize < minbundle && bundlesize != -1)
       minbundle = bundlesize;

    printf("Number of bundles = %d\n", runningnumbundles);
    printf("Average bundle size = %d bytes\n",
           totalsize / runningnumbundles);
    printf("    max bundle size = %d bytes\n", maxbundle);
    printf("    min bundle size = %d bytes\n", minbundle);
    printf("\n");
    printf("The bundle size optimal size is 128 KB or wider\n");
    printf("\n");
  }
  return 0;
}
```

SQLSTMT/SQLSBYPERF SQL Script

```
Create table Product (pid int, qty int, price float, descrip char(200));

Create table Trans (tid int, pid int, qty int);

Insert into Product values (1, 20000, 12.00, 'Turtle Cheesecake');

Create index prod_pid on Product (pid);

Create index trans_ids on Trans (pid, tid);

Insert into Product values (2, 10000, 10.00, 'Cheesecake with
Strawberries');
Insert into Product values (3, 10000, 10.00, 'Cheesecake with
Blueberries');
Insert into Product values (4, 30000, 6.00, 'Cheesecake Minis');

Insert into Product values (5, 20000, 15.00, 'Chocolate Cheesecake');
Insert into Product values (6, 35000, 13.00, 'Keylime Cheesecake');

Insert into Product values (7, 12500, 12.00, 'New York Cheesecake');

Insert into Product values (8, 25000, 13.00, 'Caramel Cheesecake');

Insert into Product values (9, 35000, 10.50, 'Cheesecake with Cherries');

Insert into product values (10, 15000, 12.00, 'Raspberry Cheesecake');
```

PERFRUN Source Code

```
/*
*  This C program will run database operations for the
*    specified number of seconds.  This program contains
*    statements which are valid only for the
*    database environment specific to the Journal Lab.
*
*
*  Syntax:
*     PERFRUN seconds
*          seconds - the number of seconds for the run
*
*
*  Compile Statement:
*      CRTSQLCI OBJ(JOLABXX/PERFRUN)
*               SRCFILE(JOLABXX/PERFRUN)
*               COMMIT(*NONE) OBJTYPE(*PGM)
*
*/


#include <stdio.h>
#include <string.h>
#include <sys/time.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{

 EXEC SQL BEGIN DECLARE SECTION;

 // variables to store the cmd line args
 unsigned long int seconds = 0;

 // variables for gettimeofday()
 struct timeval end_time;
 struct timeval cur_time;
 struct timezone timez;

 long int varpid = 1;
 long int vartid = 1;
 long int varqty = 1;

 EXEC SQL END DECLARE SECTION;

 /******************************************
  *   Parsing command line / error checking
  ******************************************/
 EXEC sql include SQLCA;


 if (argc != 2)
 {
   printf("PERFRUN seconds");
 }
 else
 {

   // get seconds from command line
   seconds = atol(argv[1]);

   // set the starting time
   gettimeofday(&end_time, &timez);

   printf("------------- PERFRUN has started running -------------\n");

   // set the end time to start time plus seconds specified
   end_time.tv_sec = end_time.tv_sec + seconds;

   // continue to loop while current time < end time
   do
   {
     // complete a database transaction
     EXEC SQL
       INSERT into Trans values (:vartid, :varpid, :varqty);
     EXEC SQL
       UPDATE Product set qty = qty - :varqty
           WHERE pid = :varpid;

     // update values for a variety of data
     vartid = vartid + 1;
     varpid = (varpid % 10) + 1;
     varqty = (varqty % 5) + 1;

     // get the current time
     gettimeofday(&cur_time, &timez);

   } while (cur_time.tv_sec < end_time.tv_sec ||
            cur_time.tv_usec <= end_time.tv_usec);

   printf("  Total running time = %u.%06u\n",
          seconds + cur_time.tv_sec - end_time.tv_sec,
          cur_time.tv_usec - end_time.tv_usec);
   printf("---------------- PERFRUN has completed ----------------\n");

 }

 return 0;
}
```

JOTIMEIT Source Code

```
/*
*  This C program outputs the time required to run the input
*     command (does not work if runtime includes change of day)
*
*  Syntax:
```

```
*     JOTIMEIT command
*          command - the command to time
*
*  Compile Statement:
         CRTBNDC PGM(JOLABXX/JOTIMEIT) SRCFILE(JOLABXX/JOTIMEIT)
*
*/

#include <stdio.h>
#include <string.h>    // strstr() call
#include <sys/time.h>

#include <qcmdexc.h>
#include <micomput.h>  // needed for decimal conversion

int main(int argc, char *argv[])
{
 // variables to store the cmd line args
 char command[1000];

 // parameters for the CL command call
 decimal(15,5) packed_length;
 int length;

 // variables for gettimeofday()
 struct timeval start_time;
 struct timeval end_time;
 struct timezone timez;

 /*******************************************
  *   Parsing command line / error checking
  ******************************************/

 if (argc != 2)
 {
   printf("JOTIMEIT command");
 }
 else
 {
   // get command line arguments
   sprintf(command, "%s", argv[1]);

   // snapshot the start time
   gettimeofday(&start_time, &timez);

   // run the command
   length = strlen(command);
   cpynv(NUM_DESCR(_T_PACKED,15,5),
         &packed_length,
         NUM_DESCR(_T_SIGNED,4,0),
         &length);
   QCMDEXC(command, packed_length);

   // get the end time
   gettimeofday(&end_time, &timez);

   // output the total runtime
   printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
   if (end_time.tv_usec > start_time.tv_usec)
   {
     printf("  Total running time = %u.%06u seconds\n",
            end_time.tv_sec - start_time.tv_sec,
            end_time.tv_usec - start_time.tv_usec);
   }
   else
   {
     printf("  Total running time = %u.%06u seconds\n",
            end_time.tv_sec - start_time.tv_sec - 1,
            100000 - start_time.tv_usec + end_time.tv_usec);
   }
   printf("\n\n");
 }
 return 0;
}
```

STRJRNLIB Source Code

```
/*
*  This C program starts journaling on all of the *FILE objects
*    in the specified library.
*
*  Syntax:
*     STRJRNLIB lib jrn jrnlib
*          lib - the library containing the objects to journal
*          jrn - the journal
*          jrnlib - the library containing the journal
*
*  Compile Statement:
         CRTBNDC PGM(JOLABXX/STRJRNLIB) SRCFILE(JOLABXX/STRJRNLIB)
*
*/

#include <stdio.h>
#include <string.h>    // strstr() call
#include <ctype.h>     // toupper() call
#include <stdlib.h>    // malloc() call

#include <qcmdexc.h>
#include <micomput.h>  // needed for decimal conversion
#include <quscrtus.h>  // create user space
#include <quslobj.h>   // list objects api
#include <qusptrus.h>  // get user space ptr
#include <qusrtvus.h>  // retrieve user space
#include <qusdltus.h>  // delete user space
#include <qusec.h>     // error structures
#include <qusgen.h>    // general user space structs

int main(int argc, char *argv[])
{
 // variables to store the cmd line args
 char lib[11];
 char jrn[11];
 char jrnlib[11];

 // counters
 int i;
 int entrynumber;
 // parameters for QUSCRTUS call
 char quserspace[21];
 Qus_EC_t *errcode;
 char errorbuffer[8];
```

```
// parameters for the QUSLOBJ call
char objandlib[21];
// parameters for QUSPTRUS
Qus_Generic_Header_0100_t *spaceptr;

// parameters for QUSRTVUS
char *objectlistptr;
char objectlist[2000];

// parameters for the CL command call
decimal(15,5) packed_length;
char command[200];
int length;

// temporary pointer
char * tempptr;

/******************************************
 *   Parsing command line / error checking
 ******************************************/

if (argc != 4)
{
  printf("STRJRNLIB lib jrn jrnlib");
}
else
{
  // get command line arguments
  sprintf(lib, "%s", argv[1]);
  sprintf(jrn, "%s", argv[2]);
  sprintf(jrnlib, "%s", argv[3]);

  // convert the lib, jrn, and jrnlib to upper case
  for (i = 0; i < 10; i++)
  {
     jrn[i] = toupper(jrn[i]);
     lib[i] = toupper(lib[i]);
     jrnlib[i] = toupper(jrnlib[i]);
  }

  // create the user space
  sprintf(qusersspace, "JOUSRSPC  %-10s", lib);
  errcode = (Qus_EC_t *) errorbuffer;
  errcode->Bytes_Provided = 0;
  QUSCRTUS(qusersspace,
           "TEMPSPACE ",
           2000,          // bytes
           " ",           // intial val
           "*ALL      ",  // authority
           "          ",  // text
           "*YES      ",  // replace
           errcode);

  // get the list of *FILE objects in the library
  sprintf(objandlib, "*ALL      %-10s", lib);
  QUSLOBJ(qusersspace,   // space object for output
          "OBJL0100", // output format
          objandlib, // object and library name
          "*FILE     ");  // object type

  // access the user space
```

```
   QUSPTRUS(quserspace,
            &spaceptr,
            errcode);

   // allocate the object list of appropriate size
   objectlistptr =
        malloc(spaceptr->Number_List_Entries * spaceptr->Size_Each_Entry +
1);
   if (objectlistptr == NULL)
   {
      printf("unable to allocate heap\n");
      return -1;
   }

   // get the object list from the user space
   QUSRTVUS(quserspace,
            spaceptr->Offset_List_Data,
            spaceptr->Number_List_Entries * spaceptr->Size_Each_Entry,
            objectlistptr);

   // data really starts on the 2nd byte
   objectlistptr += 1;
   entrynumber = 1;

   // start journaling each of the files
   while (entrynumber <= spaceptr->Number_List_Entries)
   {
     // drop in some null characters to parse out the names
     //    (if the objectname is 10 characters, a null char will
     //    not be placed at the end.  We must use a maximum of
     //    10 characters when we actually use the names)
     tempptr = strstr(objectlistptr, " ");
     if (tempptr != NULL) *tempptr = '\0';
     tempptr = strstr(objectlistptr + 10, " ");
     if (tempptr != NULL) *tempptr = '\0';

     // build and execute the start journal command
     sprintf(command,
           "STRJRNPF  FILE(%1.10s/%1.10s) JRN(%s/%s)",
           objectlistptr + 10,
           objectlistptr,
           jrnlib,
           jrn);
     length = strlen(command);
     cpynv(NUM_DESCR(_T_PACKED,15,5),
           &packed_length,
           NUM_DESCR(_T_SIGNED,4,0),
           &length);

     QCMDEXC(command, packed_length);

     // increment to the next object
     objectlistptr =
           objectlistptr + spaceptr->Size_Each_Entry;
     entrynumber += 1;
   }

   // delete the user space
   QUSDLTUS(quserspace,
            errcode);
 }
 return 0;
}
```

PARTS data file

```
PART ID    COST   MAKE             MODEL          PART
  2,001    10.00   Geo             Metro          tire
  2,002    25.00   Buick           Regal          bumper
  2,003    75.00   Ford            Taurus         exhaust system
  2,004    35.00   Dodge           Ram            windshield
  2,005    43.00   Chevy           Blazer         door
  2,006    27.00   Dodge           Ram            topper
  2,007     7.00   Buick           Regal          windshield wipers
  2,008    25.00   VW              Beetle         hood
  2,009    32.00   AMC             Hornet         motor
  2,010    14.00   Ford            Explorer       steering wheel
  2,011    30.00   Ford            Mustang        hub cap
  2,012    30.00   Plymouth        Sundance       radiator
  2,013    12.00   Volkswagen      Passat         gas cap
  2,014    78.00   Mercury         Mountaineer    axle
  2,015    55.00   Jeep            Cherokee       radiator
  2,016    34.00   Dodge           Intrepid       alternator
  2,017    48.00   Dodge           Viper          mud flag
  2,018    81.00   Chevy           Tahoe          rim
  2,019    22.00   Volkswagen      Golf           battery
      .
      .
      .
```

JODSPJRN Source Code

```
/*******************************************************************/
/*                                                                 */
/* Run DSPJRN JRN(*INTSYSJRN) JRNID(10001) FROMTIME('passed       */
/*   time').  Allows users without *ALLOBJ authority to see       */
/*   this data in the V5R2 COMMON  lab.                           */
/*                                                                 */
/*******************************************************************/
                        /*****************************************/
                        /* Program argument is the current date  */
                        /*   and time passed in a string.        */
                        /*****************************************/
             PGM        PARM(&DATE &TIME)

                        /* Passed date */
             DCL        VAR(&DATE) TYPE(*CHAR) LEN(10)
                        /* Passed time */
             DCL        VAR(&TIME) TYPE(*CHAR) LEN(10)

             MONMSG CPF0000

             DSPJRN     JRN(*INTSYSJRN) JRNID(10001) FROMTIME(&DATE &TIME)

             ENDPGM
```

MANYPARTS Source Code

```
/*
*   This program inserts the specified number of records into the
*       file specified (must have specific format for the SMAPP
*       journal Lab).
*
*   Syntax:
*       MANYPARTS file library n
*
*  Compile statement:
     CRTSQLCI OBJ(JOLABXX/MANYPARTS) SRCFILE(JOLABXX/MANYPARTS)
        SRCMBR(MANYPARTS) COMMIT(*NONE) OBJTYPE(*PGM)
**/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <qusec.h>     // Error code structures
#include <qlichgll.h>  // needed for QLICHGLL call
#include <qcmdexc.h>
#include <micomput.h> // needed for decimal conversion
#include <ctype.h>     // toupper() call

 EXEC sql include SQLCA;

int main(int argc, char *argv[])
{
 long int i;
 int num;
 char file[11];
 char lib[11];
 char statement[200];


 if (argc != 4)
 {
   printf("MANYPARTS file lib num \n ");
 }
 else
 {
   sprintf(file, "%-10s", argv[1]);
   sprintf(lib, "%-10s", argv[2]);
   num = atoi(argv[3]);

   // convert the dtaq and library to upper case
   for (i = 0; i < 10; i++)
   {
      file[i] = toupper(file[i]);
      lib[i] = toupper(lib[i]);
   }

   sprintf(statement,
      "INSERT into %s/%s values (?, 1.00, 'car', 'make', 'part', 2002)",
      lib, file);
   EXEC SQL
     PREPARE actstmt FROM :statement;
   for (i = 10000; i < (10000 + num); i++)
   {
     EXEC SQL
       EXECUTE actstmt USING :i;
   }
 }
 return 0;
}
```

SQLSTMT/SQLSPT SQL Script

```
Create table Customer (cid int, Name char(30));
Create table Hotel (cid int, H_Name char(30));
Create table Flight (cid int, Airline char(30), depart date);
Create table Car_rental (cid int, R_Comp char(30), pickup date);
```

PARSEJOPEX Source Code

```
/*
*
*   This program parses the Journal PEX Trace point events in
*       QAYUSRDFN and puts them into a nicely formatted table
*
*
*
*  Syntax:  PARSEJOPEX library
*               library = library which contains results
*                           also used as output file location
*
*  Compile statement:
     CRTSQLCI OBJ(JOLABXX/PARSEJOPEX) SRCFILE(JOLAB/PARSEJOPEX)
        SRCMBR(PARSEJOPEX) COMMIT(*NONE) OBJTYPE(*PGM)
**/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <ctype.h>     // toupper() call

#include <qusec.h>
#include <qlichgll.h>
#include <qcmdexc.h>
#include <micomput.h>

EXEC sql include SQLCA;

int main(int argc, char *argv[])
{

EXEC SQL BEGIN DECLARE SECTION;
 /* command line parameters */
 char lib[11];

 /* host vars */
 long int cnt = 0;
 char dta[300];
 short int typ = 0;
 short int styp = 0;

 /* a counter */
 long int i;
 int j;

 /* for QLICHGLL call */
 char current[11];
 char same[11];
 long int num_user_libs = -1;
 Qus_EC_t *err_code;
 char errorbuffer[512];

 /* for QCMDEXC call */
 decimal(15,5) packed_length;
 int length;
 char command[200];

 char statement[200];

 /* data */
 char port[16];
EXEC SQL END DECLARE SECTION;

 if (argc != 2)
 {
   printf("PARSEJOPEX lib\n");
 }
 else
 {
   sprintf(lib, "%s", argv[1]);

   /* convert the library to upper case */
   for (i = 0; i < 10; i++)
   {
      lib[i] = toupper(lib[i]);
   }

   /* set the current library */
   sprintf(current, "%-10s", lib);
   strcpy(same, "*SAME      ");
   err_code = (Qus_EC_t *) errorbuffer;
   err_code->Bytes_Provided = 0;
   QLICHGLL(current,
            same,
            same,
            same,
            num_user_libs,
            err_code);

   /* create the output table */
   EXEC SQL
       CREATE TABLE JOPEXOUT
      (record int,
       type int,
       subtype int,
       portaddr char(16),
       accpathaddr char(16),
       objectaddr char(16),
       recrasetting char(16),
       recraobjects char(8),
       smapptrans char(2),
       smappreason char(2),
       cycleid char(16),
       curSeqNum char(16),
       curSyncPoint char(16),
       writes char(8),
       permWrites char(8),
       syncDbReads char(8),
       syncNonDbReads char(8),
       syncDbWrites char(8),
       syncNonDbWrites char(8),
       asyncDbReads char(8),
       asyncNonDbReads char(8),
       asyncDbWrites char(8),
       asyncNonDbWrites char(8),
       accessGroupFaults char(8),
       ioPendingWaits char(8),
       synIoWaits char(8),
       pageFaults char(8),
       seizewt char(16),
       mJOCommitOps char(8),
       mJOSecCommitOps char(8),
       mJODecommOps char(8),
       mJOSecDecommOps char(8),
       mJOJrnWrts char(8),
       mJONonSMAPP char(8),
       mJOUser char(8),
       mJOSystem char(8),
       mJOFixedBytes char(16),
       mJODepFixedBytes char(16),
       mJOTransient char(16),
       mJODepTransient char(16),
       mJONaptime char(16),
       mJONaps char(8),
       mJOEval_Messages char(8),
       mJOTriggered_Evals char(8),
       mJOBuildtime_Ests char(8),
       queuingWaitTime char(16));

   /* copy the QAYPEUSRDF and QAYPETIDX table */
   /* copy the table because a table of the same name exists in
    *   QSYS which will be found prior to the one in the
    *   specified library due to the library list.  A new name
    *   will allow this one to be found without specifying
    *   a library.   This is due to the fact that I do not know
    *   how to use a host variable in a dynamic statement which
    *   would allow me to specify the library name passed in.
    */
   sprintf(command,
           "CPYF %s/QAYPEUSRDF %s/USRDFCOPY CRTFILE(*YES)",
           lib, lib);
   length = strlen(command);
   cpynv(NUM_DESCR(_T_PACKED,15,5), &packed_length,
         NUM_DESCR(_T_SIGNED,4,0), &length);
   QCMDEXC(command, packed_length);

   sprintf(command,
           "CPYF %s/QAYPETIDX %s/TIDXCOPY CRTFILE(*YES)",
           lib, lib);
```

```
length = strlen(command);
cpynv(NUM_DESCR(_T_PACKED,15,5), &packed_length,
      NUM_DESCR(_T_SIGNED,4,0), &length);
QCMDEXC(command, packed_length);

/* get the number of events to parse  */
EXEC SQL
   select max(QRECN) into :cnt from USRDFCOPY;

for (i = 1; i <= cnt; i++)
{
  /* get a row of data to parse  */
  EXEC SQL
     select QUSDTA into :dta from USRDFCOPY where QRECN = :i;
 EXEC SQL
 INSERT into JOPEXOUT
   (record,
    portaddr,
    accpathaddr,
    objectaddr,
    recrasetting,
    recraobjects,
    smapptrans,
    smappreason,
    cycleid,
    curSeqNum,
    curSyncPoint,
    writes,
    permWrites,
    syncDbReads,
    syncNonDbReads,
    syncDbWrites,
    syncNonDbWrites,
    asyncDbReads,
    asyncNonDbReads,
    asyncDbWrites,
    asyncNonDbWrites,
    accessGroupFaults,
    ioPendingWaits,
    synIoWaits,
    pageFaults,
    seizewt,
    mJOCommitOps,
    mJOSecCommitOps,
    mJODecommOps,
    mJOSecDecommOps,
    mJOJrnWrts,
    mJONonSMAPP,
    mJOUser,
    mJOSystem,
    mJOFixedBytes,
    mJODepFixedBytes,
    mJOTransient,
    mJODepTransient,
    mJONaptime,
    mJONaps,
    mJOEval_Messages,
    mJOTriggered_Evals,
    mJOBuildtime_Ests,
    queuingWaitTime)
    select QRECN,
    HEX(SUBSTR(QUSDTA, 1, 8)) portaddr,
    HEX(SUBSTR(QUSDTA, 9, 8)) accpathaddr,
    HEX(SUBSTR(QUSDTA, 17, 8)) objectaddr,
    HEX(SUBSTR(QUSDTA, 25, 8)) recrasetting,
    HEX(SUBSTR(QUSDTA, 33, 4)) recraobjects,
    HEX(SUBSTR(QUSDTA, 37, 1)) smapptrans,
    HEX(SUBSTR(QUSDTA, 38, 1)) smappreason,

    HEX(SUBSTR(QUSDTA, 41, 8)) cycleid,
    HEX(SUBSTR(QUSDTA, 49, 8)) curSeqNum,
    HEX(SUBSTR(QUSDTA, 57, 8)) curSyncPoint,

    HEX(SUBSTR(QUSDTA, 65, 4)) writes,
    HEX(SUBSTR(QUSDTA, 69, 4)) permWrites,
    HEX(SUBSTR(QUSDTA, 73, 4)) syncDbReads,
    HEX(SUBSTR(QUSDTA, 77, 4)) syncNonDbReads,
    HEX(SUBSTR(QUSDTA, 81, 4)) syncDbWrites,
    HEX(SUBSTR(QUSDTA, 85, 4)) syncNonDbWrites,
    HEX(SUBSTR(QUSDTA, 89, 4)) asyncDbReads,
    HEX(SUBSTR(QUSDTA, 93, 4)) asyncNonDbReads,
    HEX(SUBSTR(QUSDTA, 97, 4)) asyncDbWrites,
    HEX(SUBSTR(QUSDTA, 101, 4)) asyncNonDbWrites,
    HEX(SUBSTR(QUSDTA, 105, 4)) accessGroupFaults,
    HEX(SUBSTR(QUSDTA, 109, 4)) ioPendingWaits,
    HEX(SUBSTR(QUSDTA, 113, 4)) synIoWaits,
    HEX(SUBSTR(QUSDTA, 117, 4)) pageFaults,
    HEX(SUBSTR(QUSDTA, 121, 8)) seizewt,

    HEX(SUBSTR(QUSDTA, 129, 4)) mJOCommitOps,
    HEX(SUBSTR(QUSDTA, 133, 4)) mJOSecCommitOps,
    HEX(SUBSTR(QUSDTA, 137, 4)) mJODecommOps,
    HEX(SUBSTR(QUSDTA, 141, 4)) mJOSecDecommOps,
    HEX(SUBSTR(QUSDTA, 145, 4)) mJOJrnWrts,
    HEX(SUBSTR(QUSDTA, 149, 4)) mJONonSMAPP,
    HEX(SUBSTR(QUSDTA, 153, 4)) mJOUser,
    HEX(SUBSTR(QUSDTA, 157, 4)) mJOSystem,
    HEX(SUBSTR(QUSDTA, 161, 8)) mJOFixedBytes,
    HEX(SUBSTR(QUSDTA, 169, 8)) mJODepFixedBytes,
    HEX(SUBSTR(QUSDTA, 177, 8)) mJOTransient,
    HEX(SUBSTR(QUSDTA, 185, 8)) mJODepTransient,
    HEX(SUBSTR(QUSDTA, 193, 8)) mJONaptime,
    HEX(SUBSTR(QUSDTA, 201, 4)) mJONaps,

    HEX(SUBSTR(QUSDTA, 205, 4)) mJOEval_Messages,
    HEX(SUBSTR(QUSDTA, 209, 4)) mJOTriggered_Evals,
    HEX(SUBSTR(QUSDTA, 213, 4)) mJOBuildtime_Ests,

    HEX(SUBSTR(QUSDTA, 217, 8)) queuingWaitTime

    from USRDFCOPY where QRECN = :i;

 EXEC SQL
    SELECT QTITY, QTISTY
      into :typ, :styp
       from TIDXCOPY
    where QRECN = :i;

 EXEC SQL
    UPDATE JOPEXOUT
        SET type = :typ, subtype = :styp
    where record = :i;
   }

   /* clean up the temp files */
   sprintf(command,
           "DLTF TIDXCOPY",
           lib, lib);
   length = strlen(command);
   cpynv(NUM_DESCR(_T_PACKED,15,5), &packed_length,
         NUM_DESCR(_T_SIGNED,4,0), &length);
   QCMDEXC(command, packed_length);

   /* clean up the temp files */
   sprintf(command,
           "DLTF USRDFCOPY",
           lib, lib);
   length = strlen(command);
   cpynv(NUM_DESCR(_T_PACKED,15,5), &packed_length,
         NUM_DESCR(_T_SIGNED,4,0), &length);
   QCMDEXC(command, packed_length);

   /* set the current library to nothing */
   sprintf(current, "*CRTDFT   ");
   strcpy(same, "*SAME      ");
   err_code = (Qus_EC_t *) errorbuffer;
   err_code->Bytes_Provided = 0;
   QLICHGLL(current,
            same,
            same,
            same,
            num_user_libs,
            err_code);
 }

 return 0;
}
```