Agenda

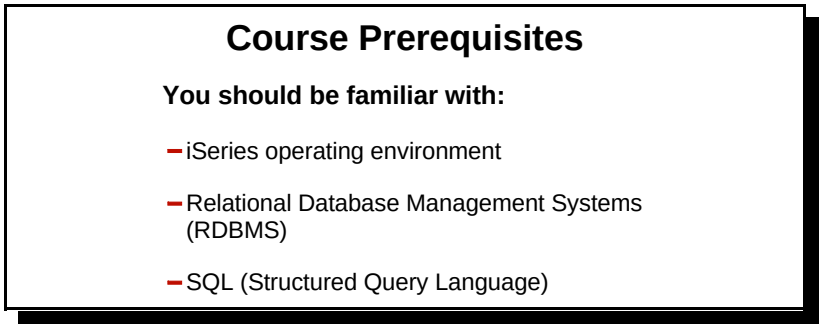
Author Biographical Information
Course Prerequisites
Overview
Agenda: Static and Dynamic SQL
SQL Interfaces
Static SQL
Dynamic SQL
Dynamic SQL Interfaces
SQL Access Plans and ODPs
Access Plans — Static SQL View
Access Plans — Plan Contents
Access Plans — Dynamic SQL View
Access Plans — Extended Dynamic SQL View
OPENing the Access Plan
Reasons for Rebuilding the Access Plan
Reasons for Rebuilding the Access Plan (continued)
Reasons for building an Access Plan — an example
Access Plan Rebuild Considerations
SQE Plan Cache
SQE Plan Cache
Access Plan to an Open Data Path (ODP)
ODPs
ODPs “in Action”
OPEN Optimization — OPENs
OPEN Optimization — Reusable ODPs
Reusing the ODP Steps
Reusing the ODP Example
OPEN Optimization — Reusable ODP Example
Reusable ODP Tips and Techniques
OPEN Optimization — Reuse Roadblocks
OPEN Optimization — Reuse Roadblocks (continued)
OPEN Optimization — Reuse Roadblocks (continued)
OPEN Optimization — UPDATE WHERE CURRENT OF Reuse
OPEN Optimization — Reuse Considerations
OPEN Optimization — Actions that Delete ODPs
OPEN Optimization
Dynamic and Extended Dynamic SQL

Author Biographical Information

Kent Milligan, DB2 UDB Technology Specialist
IBM eServer Solutions Enablement

Kent Milligan is a DB2 UDB for iSeries Technology Specialist in the IBM eServer Solutions Enablement organization. Kent spent the first eight years of his IBM career as a member of the DB2 development group in Rochester, MN. He speaks and writes regularly on relational database topics. He can be reached at: kmill@us.ibm.com.

.

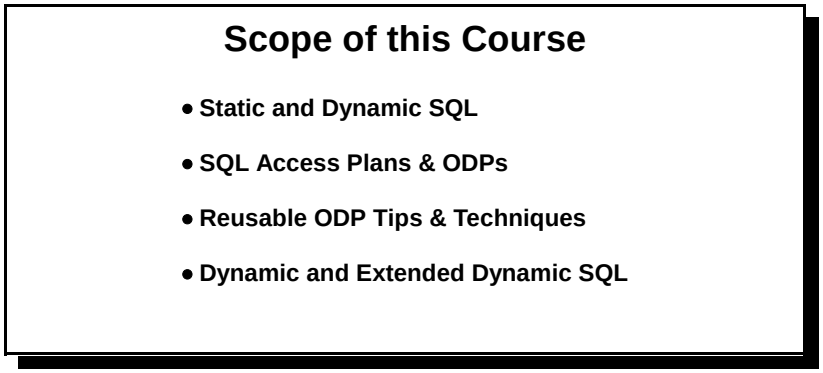


Course Prerequisites

Hello, and welcome to this updated SQL Performance Basics online course! Let's review a few topics which we assume you are already familiar with. For instance, you should have a basic understanding of the IBM® eServer™ iSeries™ operating environment. We also assume you are already familiar with the basics of Relational Database Management Systems (RDBMS) and Structured Query Language (SQL).

If, at any point, you wish to jump back to a previous section of this course, simply click on the "Agenda" hotlink in the navigation bar, and then click on the segment you want to review. When you are finished with reviewing that segment, you will be given the opportunity to return to the prior segment of the course you were viewing.

You will occasionally be quizzed on covered material to help you determine your level of understanding before moving onto the next segment. Then, at the end of the course, you will be asked to take a final test regarding the material.



Overview

This online course has been designed to provide you with an overview of the DB2® UDB for iSeries SQL implementation. You will become familiar with: static and dynamic SQL, SQL access plans and open data paths (ODPs), tips and techniques for utilizing reusable ODPs, and dynamic and extended dynamic SQL.

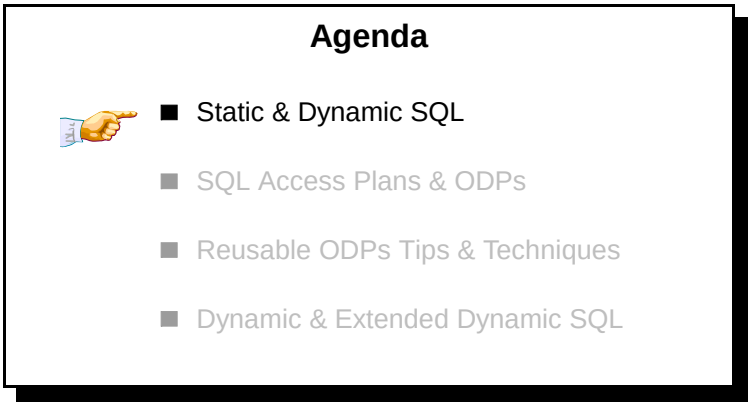
First, the components of SQL interfaces are examined from the bottom up to provide a better understanding of how a programmer builds up, or out, across the network, to the client query program interfaces. Each building block of the database architecture is studied. The elements of static and dynamic SQL are reviewed, along with DB2 UDB for iSeries interfaces that utilize SQL.

SQL access plans and ODPs are explained, along with considerations of rebuilding or reusing ODPs. Reasons for creating or reusing ODPs are examined. The elements of access plans are considered through static and dynamic SQL views. Examples of how ODPs are used in “real world situations” are presented through mock case studies.

Handy tips and techniques for reusing ODPs are also covered, and potential roadblocks that force ODP creations are explained by examples.

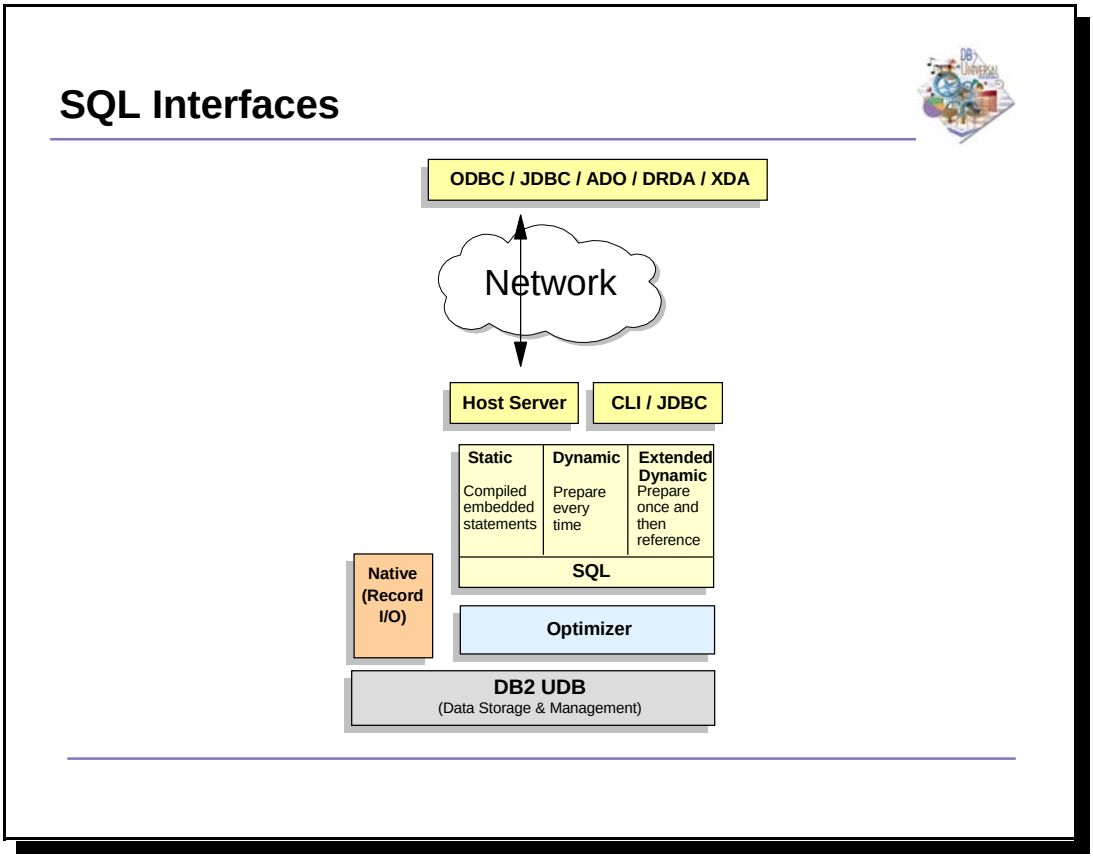
The course concludes with examinations of APIs and XDA APIs, and their impact on the DB2 UDB for iSeries SQL engine.

So, let’s get started . . .



Agenda: Static and Dynamic SQL

First, let's take a look at static and dynamic SQL.

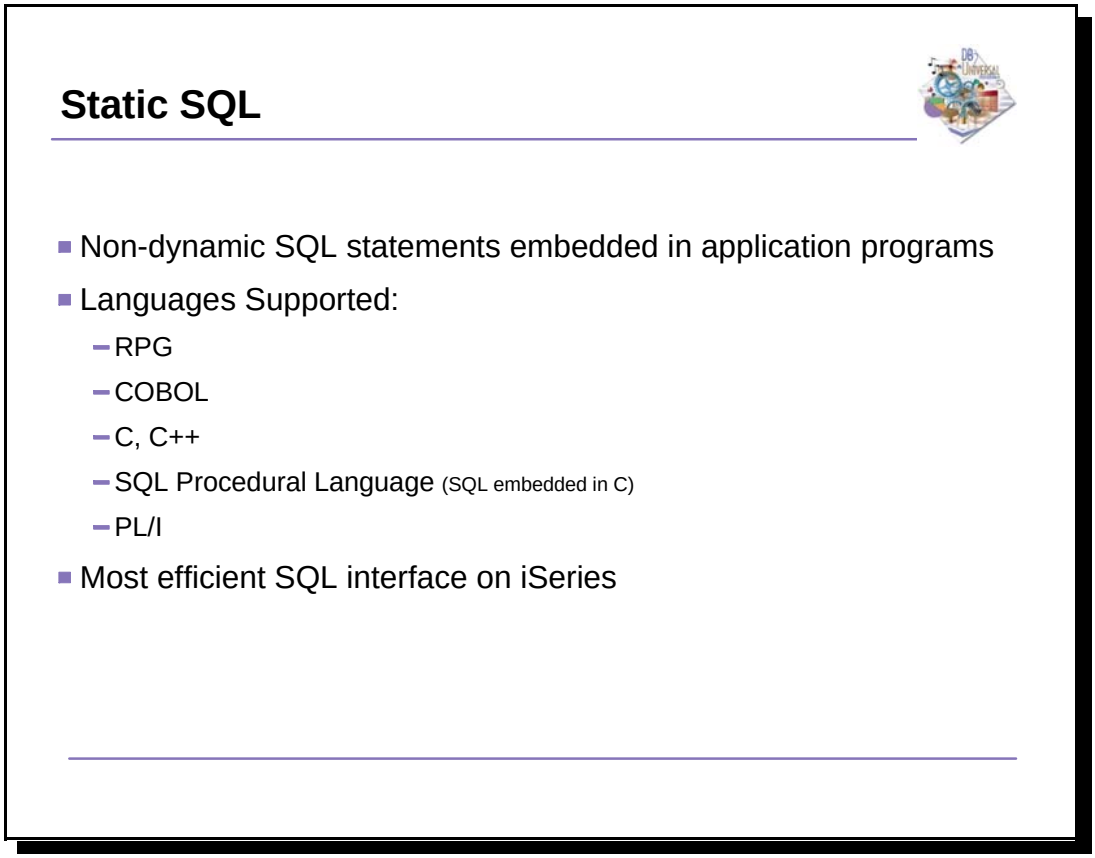


SQL Interfaces

Here are the components of SQL interfaces. We will cover all these components within this course, but will do so from the bottom up. This approach will give you a better understanding of how you build up, or out, across the network, to the client SQL program interfaces.

This is a high level picture of the DB2 UDB for iSeries architecture and illustrates where the optimization occurs.

Let's take the SQL blocks of the DB2 UDB for iSeries architecture one step at a time.

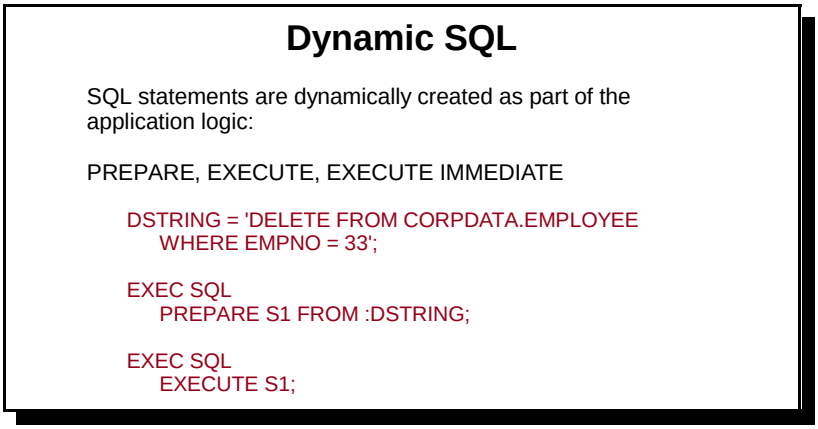


Static SQL

The term *"embedded SQL"* is also known as *"static SQL"* when referring to SQL as it operates on DB2 UDB for iSeries. Actually, static SQL statements on an iSeries server are more logically referred to as "non-dynamic SQL statements that are embedded in a program." This distinction is important to understand because high level languages (HLLs) on iSeries servers support the embedding of SQL statements (such as PREPARE, EXECUTE, and EXECUTE IMMEDIATE) — much in the same way as you can embed an INSERT or UPDATE statement into an HLL program.

That is to say, SQL precompilers allow you to embed SQL within most of the popular iSeries programming languages — RPG, COBOL, C and C++, as well as the SQL Stored Procedure Language.

Embedded SQL is the most efficient SQL interface to use on an iSeries server since DB2 UDB for iSeries stores the SQL access plan in the program objects. We will see some pictures portraying these differences in a couple charts.



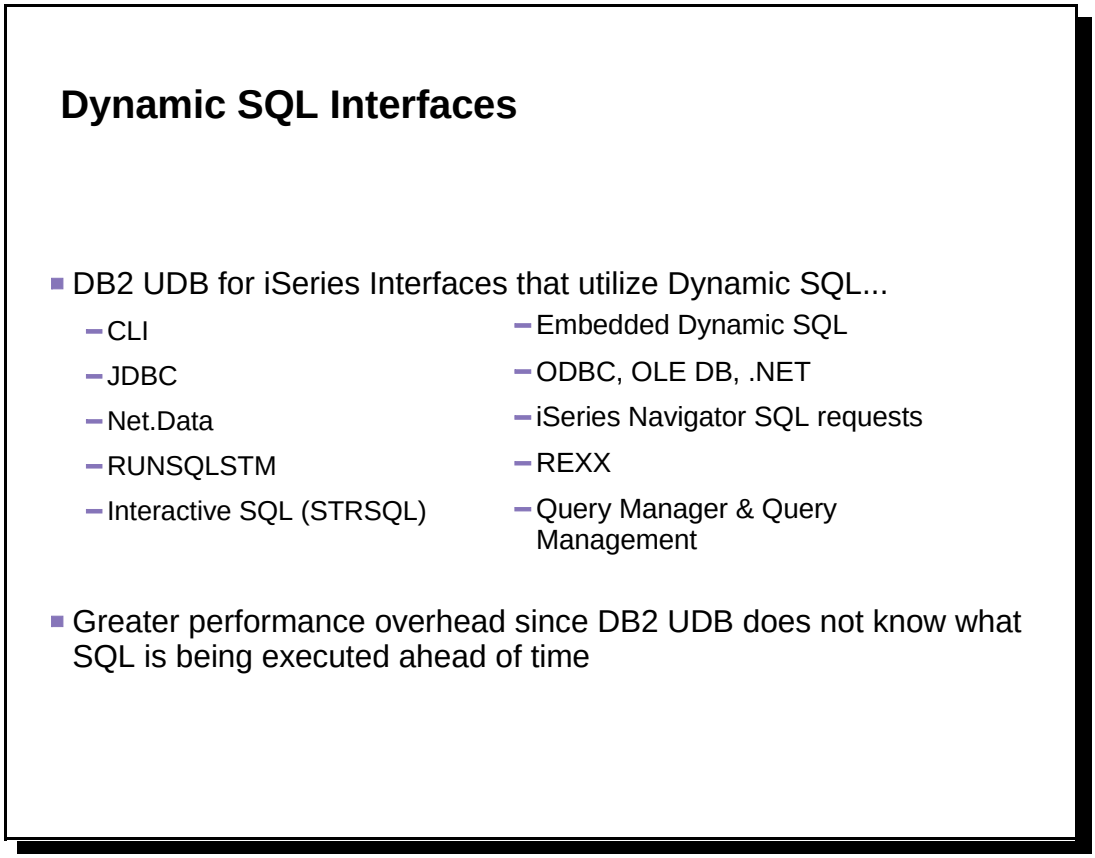
Dynamic SQL

Since the dynamic SQL interface constructs SQL statements on the fly, the optimizer cannot define an SQL access path ahead of time.

Often, Dynamic SQL is necessary because you will not know exactly what type of SQL statement needs to be executed until the user selects specific check boxes or chooses a particular menu option.

Normally, the user interface causes multiple parts of a dynamic SQL statement to be concatenated into one final SQL statement text. Typical SQL commands that would be used to build dynamic SQL statements include: PREPARE, EXECUTE, and EXECUTE IMMEDIATELY.

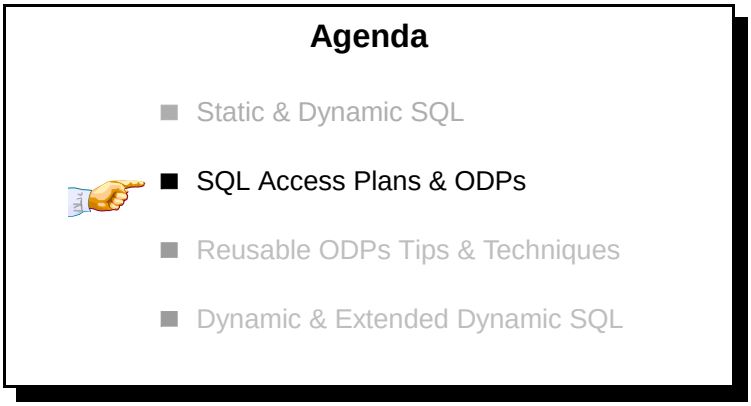
Notice the code in the graphic shows how an SQL statement is dynamically built.



Dynamic SQL Interfaces

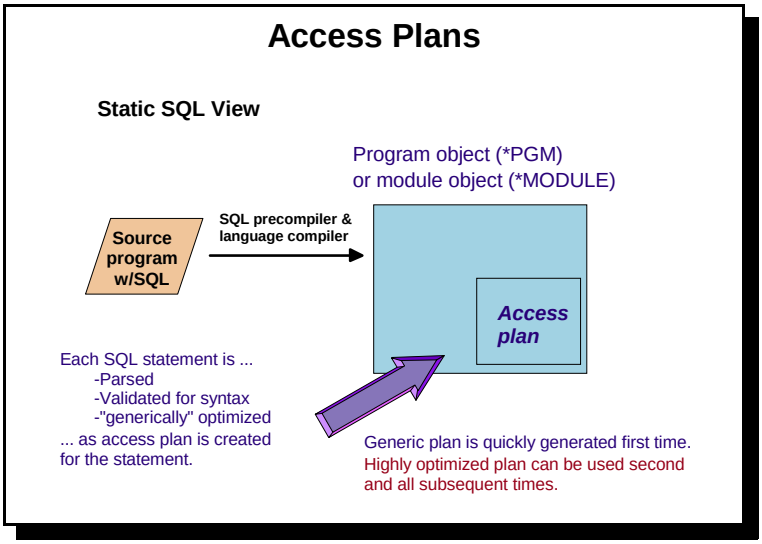
This graphic shows the most commonly used DB2 UDB for iSeries interfaces that utilize dynamic SQL.

Note that, since DB2 UDB for iSeries does not know what SQL request will be executed until it is called by the client, these dynamic interfaces require greater performance overhead from an iSeries server.



SQL Access Plans and ODPs

This course is generally focused on ways you can improve SQL performance in cases where one SQL statement will be executed numerous times for the same job or request. This is typical of OLTP and ERP environments more than it is for business intelligence (BI) and decision support solutions (DSS). In BI and DSS, end users run a report just once, or they are in a mode where they run the report once a week, once a month, etc.



Access Plans — Static SQL View

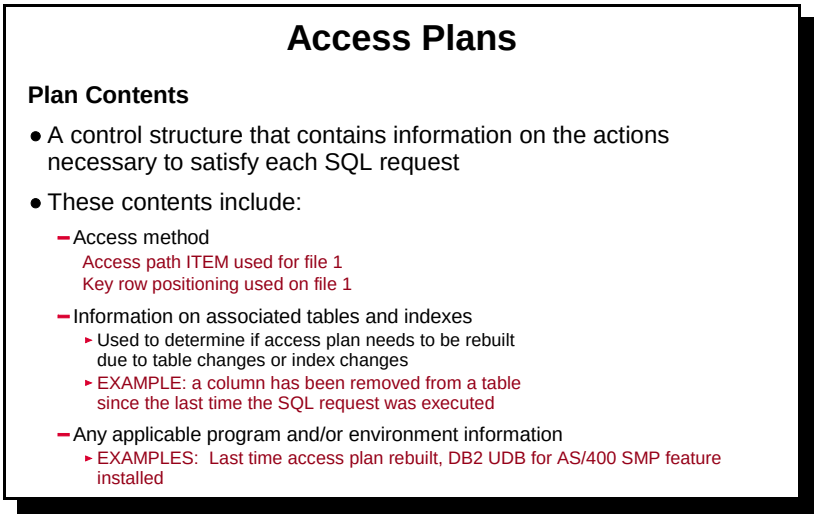
Here's a pictorial representation of static SQL.

The precompiler validates and parses the SQL requests in the HLL program. An access plan is then created for each SQL statement and is stored in the program object or module object. (The contents of an access plan is explained on the next chart)

Static embedded SQL is fast because the access plan is stored in this static program object. When the program is run, the validation and access-plan creation steps can be skipped — thus, improving performance and making the entire query effort much faster.

Since all optimization parameters are not known prior to actual execution, the optimizer creates only a generic optimization plan during precompile. However, the plan receives detailed analysis on the first execution of the statement, and then the optimizer updates the original access plan. So, the first time you run the statements in an SQL program, things will be a little slower than during each subsequent running of those statements.

NOTE: Keep this last point in mind if you are conducting benchmark activities.



Access Plans — Plan Contents

An access plan is the control structure that contains all of the information regarding the actions that will be required to satisfy each embedded SQL request.

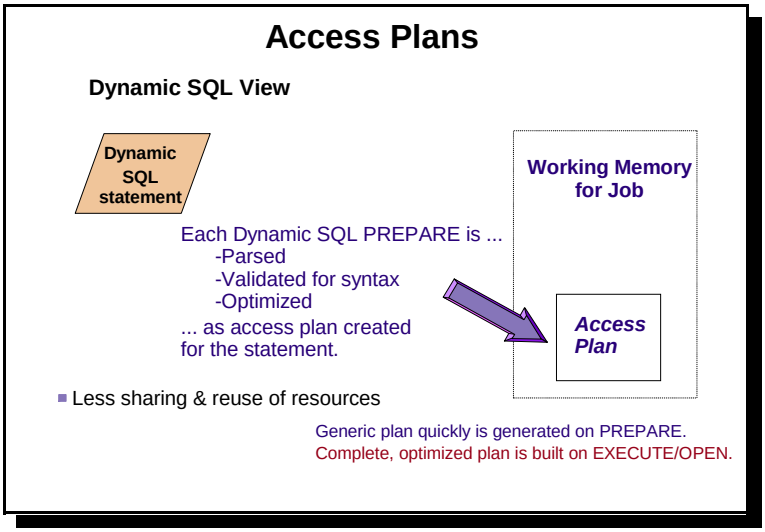
These contents include three components:

* First, an access method. The access method is the method chosen by the query optimizer to implement the SQL request (i.e., table scan versus index scan). The iSeries optimizer chooses the access plan with the lowest cost since it is a cost-based optimizer.

* Second, information that can be found in associated tables and indexes. This information is needed to determine if the access plan must be rebuilt due to any table changes or index changes since the last time the SQL request was executed. For example, a column may have been recently removed from a table, thereby effecting the access path.

* Third, any applicable program and/or environment information. An example of program environment information would be data regarding the last time the access plan was rebuilt. A perfect example of environment information would be a situation where the DB2 UDB for iSeries SMP feature has been recently installed, thereby offering additional optimization methods for SQL requests.

NOTE: Access plan contents are the same for both static and dynamic SQL requests.

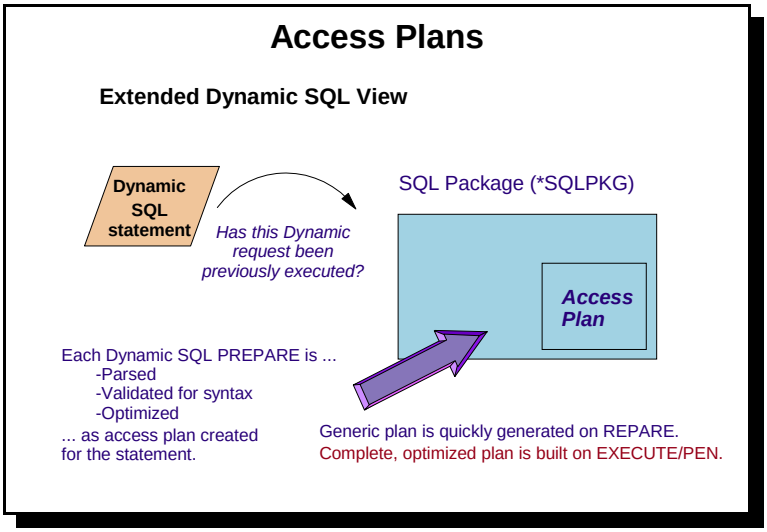


Access Plans — Dynamic SQL View

In the Dynamic SQL view, there are no sharing of resources. Accessed information must be "relearned" by every job and every user. There is also no static system object where the access plan can be stored.

For Dynamic SQL, a generic access plan is quickly generated at PREPARE time. (During the PREPARE, the SQL is also parsed and validated for syntax.) The complete, highly-optimized plan is generated at OPEN or EXECUTE time.

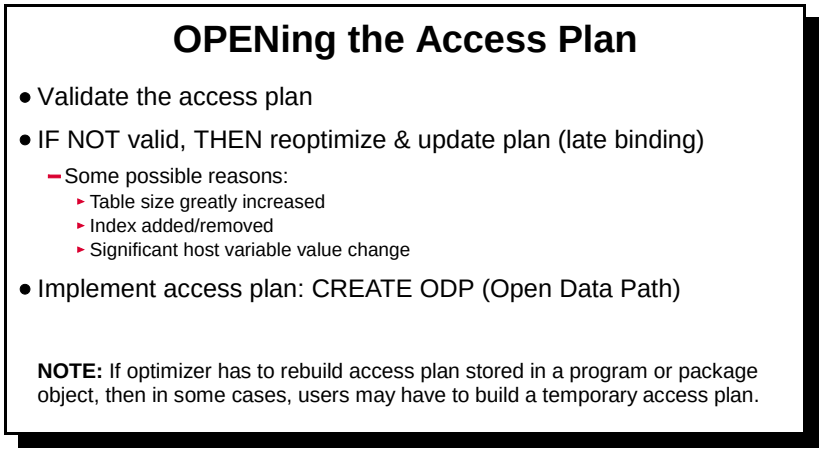
A system-wide statement cache attempts to help with the reuse of access plans for Dynamic SQL requests.



Access Plans — Extended Dynamic SQL View

The Extended Dynamic SQL view was introduced on the iSeries platform for some SQL interfaces to improve the performance of Dynamic SQL requests. In these cases, a permanent system object (SQL package) is used to store the access plan — thus allowing Dynamic SQL access plans to be shared across users and to remain intact after a job or connections ends.

For those of you who may have experience with Distributed Relational Database Architecture (DRDA) on iSeries, please note that any discussion of SQL packages during this online course does not relate to DRDA's usage of packages.
For further information regarding DRDA on iSeries capabilities, check the *DRDA: Application Programming Guide (SC26-4773)* located at the iSeries Information Center. (See the Links section of this course for the URL.)



OPENing the Access Plan

Once a valid access plan exists, here are the steps that DB2 UDB for iSeries goes through to implement the associated SQL request. The plan is a blueprint for how the data will be accessed and the ODP (Open Data Path) is the pipe that gets the data in and out of your database objects.

During the process of OPENing the access plan, ODP helps you to understand that no bind commands or statements exist on iSeries servers. DB2 UDB for iSeries does the binding automatically. Even more impressive (and convenient), the server automatically rebuilds and rebinds the plan if the environment has changed. That is, if a new index is created to improve performance, then DB2 UDB for iSeries automatically recognizes the new index and rebuilds the access plan to take advantage of the new index.

Reasons for Rebuilding the Access Plan

Message ID - CPI4323

Message . . . . : The OS/400 Query access plan has been rebuilt.
Cause . . . . . : The access plan was rebuilt for reason code &13. The reason codes and their meanings follow:

1 - A file or member is not the same object as the one referred to in the access plan. Some reasons they could be:
- Object was deleted and re-created or restored.
- Library list was changed.
- Object was renamed or moved.
- Object was overridden (OVRDBF CL command) to a different object.
- This is the first run of this query after the object containing the query has been restored.

2 - Access plan was using a reusable Open Data Path (ODP), and the optimizer chose to use a non-reusable ODP.
3 - Access plan was using a non-reusable Open Data Path (ODP) and the optimizer chose to use a reusable ODP.
4 - The number of records in member &3 of file &1 in library &2 has changed by more than 10%.
5 - A new access path exists over member &6 of file &4 in library &5.
6 - An access path over member &9 of file &7 in library &8 that was used for this access plan no longer exists or is no longer valid.
7 - OS/400 Query requires the access plan to be rebuilt because of system programming changes.
8 - The CCSID (Coded Character Set Identifier) of the current job is different than the CCSID used in the access plan.
9 - The value of one of the following is different in the current job: date format, date separator, time format, or time separator.
10 - The sort sequence table specified has changed.
11 - The size of the storage pool, or paging option of the storage pool has changed and estimated runtime is less than 2 seconds
- ►CQE optimizer only rebuilds plan when there has been a 2X change in memory pool size and runtime estimate less than 2 seconds
- ►SQE optimizer only rebuilds plan with a 2X change in memory pool size

12 - The system feature DB2 Symmetric Multiprocessing has either been installed or removed.
13 - The value of the degree query attribute has changed either by the CHGSYSVAL or CHGQRYA CL commands.
14 - A view is either being opened by a high level language open, or view is being materialized.

If the reason code is 4, 5, or 6 and the file specified in the reason code explanation is a logical file,
then member &12 of physical file &10 in library &11 is the file with the specified change.

Reasons for Rebuilding the Access Plan

Don’t let this chart throw you. Stay with us! This is just the iSeries system message that's signaled every time an access plan is rebuilt. All the various reasons that may cause the rebuild are listed in the second-level text of the message. Here are some of the common reasons that an access plan must be rebuilt:

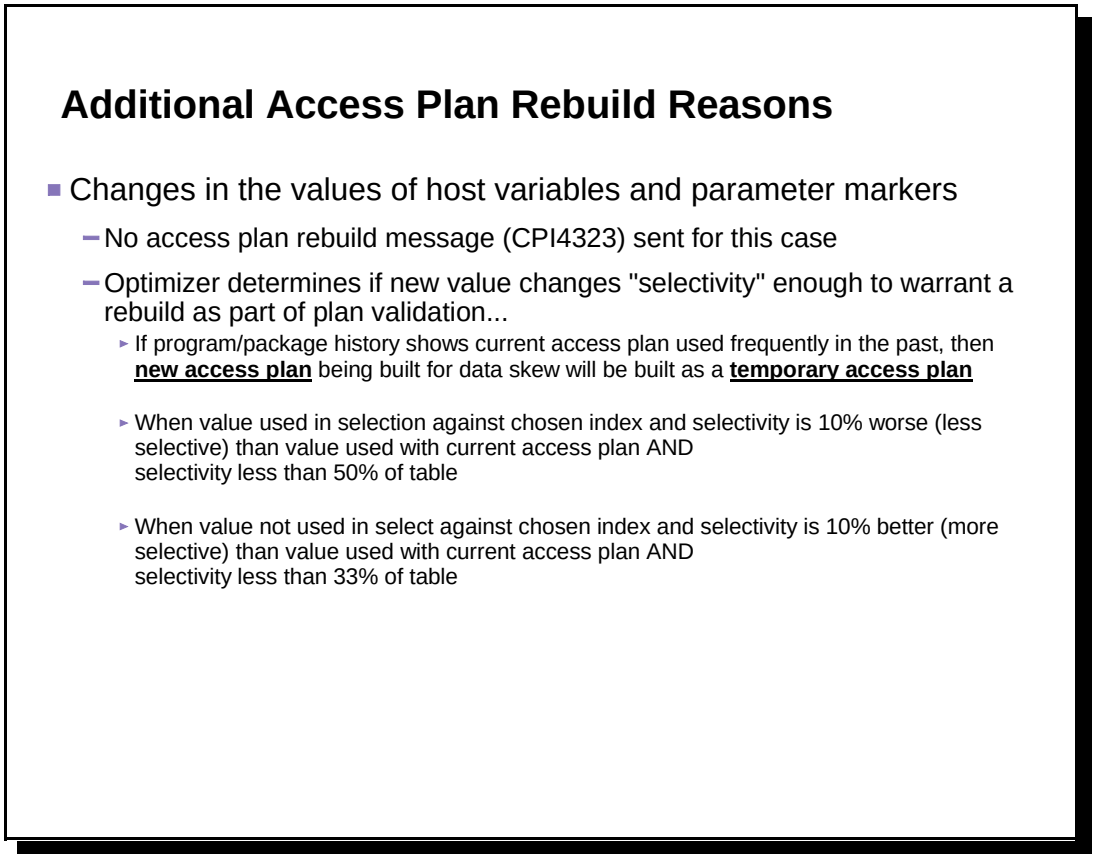
* The table name is unqualified and the library list has changed. The number of rows in the table being queried has changed by more than 10%.

* The table being queried has a new index, or a previously used index no longer exists.

* System programming changes require a new access plan. For example, the installation of a new OS/400® release or Database Group PTF.

* The amount of main memory available to the SQL request has changed. (Not all main memory adjustments will cause an access plan rebuild; the database only adjusts to significant main memory changes).

Now, that wasn’t too bad ... was it? We will talk further about these reasons for rebuilding the access plan next.



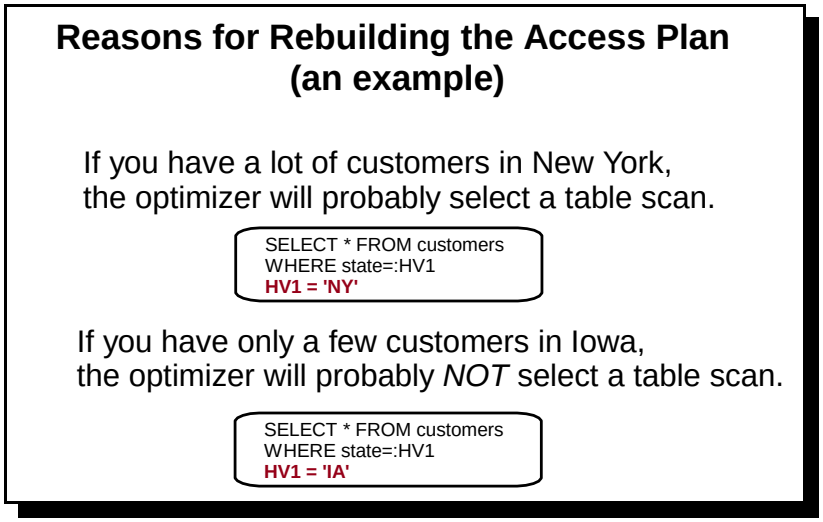
Reasons for Rebuilding the Access Plan (continued)

Changes in the values of host variables and parameter markers will force the rebuilding of the access plan. But, you should know that no access plan rebuild message (message ID CPI4323, displayed in a previous chart) is sent to the administrator or the user if this is the reason for the rebuild.

The optimizer determines if the new value changes "selectivity" enough to warrant a rebuild as part of the plan validation. Here are two examples of host variable and parameter marker changes that will trigger an access plan rebuild:

* When the value is used in the “select against chosen” index, *AND* selectivity is 10% less (less selective) than the value used with current access plan, *AND* selectivity is less than 50% of the table; the access plan is rebuilt.

* When the value is not used in the “select against chosen” index, *AND* selectivity is 10% better (that is, more selective) than the value used with the current access plan, *AND* selectivity is less than 33% of the table; the access plan is rebuilt.



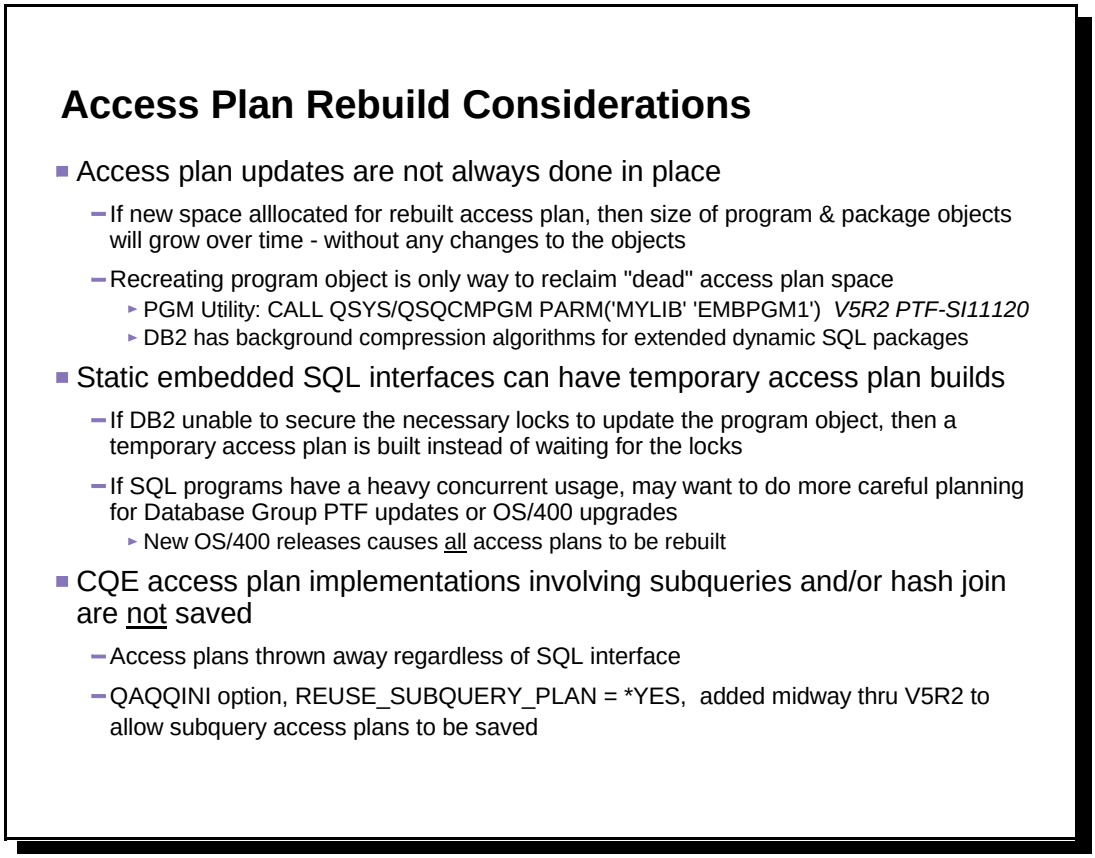
Reasons for building an Access Plan — an example

Let's use the following example to illustrate our point.

Your server supports a national company on the East Coast. The first time the SQL statement is run, the program presents a host variable value of "NY" (New York). The optimizer is very likely to choose a table scan because one-half of your dynamic customers live in New York City.

The next time the SQL statement is run, someone is doing an analysis on the customers in "IA" (Iowa). This could very well only return one to two percent of your customers. Is it a good idea to use table scans to only select two percent of the rows? Probably not.

As the previous chart mentions, DB2 keeps some execution history statistics to determine which access method is used most often. So in this case, if the historical statistics show that the search value of "NY" is searched for more often than smaller states like "IA," the access plan will not be rebuilt. Instead, a temporary access plan will be built in memory for the search values and access methods that are executed less frequently.



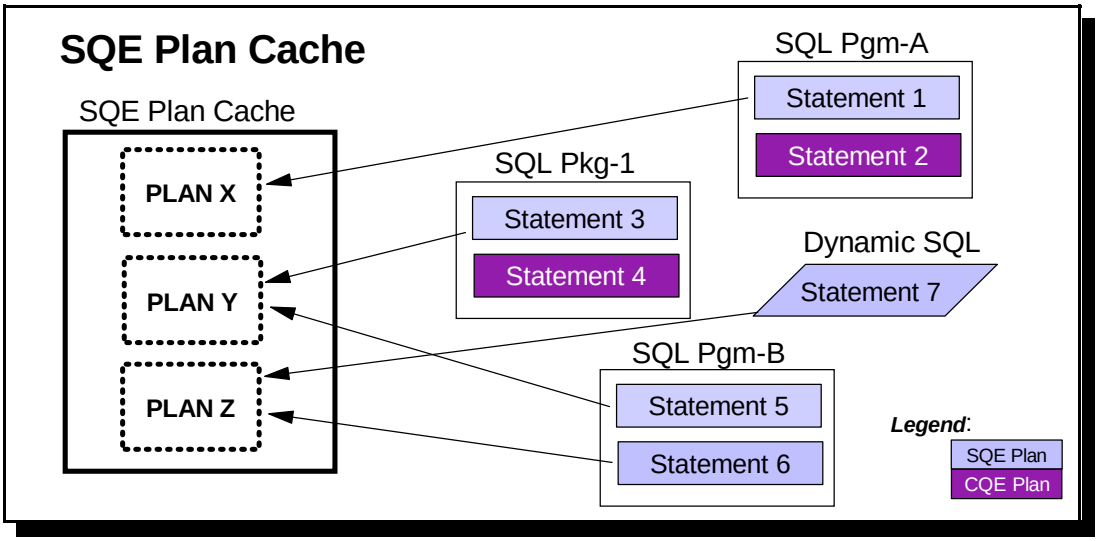
Access Plan Rebuild Considerations

Sometimes an access plan cannot be rebuilt in-place—instead, new space must be allocated for the rebuilt plan. This behavior means a program or package object can grow in size with no changes being made to the application. If the program or package object size limit is hit, than the "dead" space from old access plans is used by recreating the program or package object. If the application source code is not available for recreating the program object, a utility is available from IBM Support to reclaim the dead space.

Even though static embedded SQL and extended Dynamic SQL have a permanent container for storing access plans, there are times when temporary access plans will be built. When an access plan is updated, DB2 UDB for iSeries must serialize access to the program (or package) object by acquiring locks. If the object is already locked, DB2 will build a temporary access plan instead of waiting for the lock to be released.

There are a couple of cases where CQE (Classic Query Engine) always builds a temporary access plan. If the query request is implemented using the hash join algorithm or includes a subquery, then the temporary access plans are always created by CQE. An option is available to allow access plans referencing a subquery to be saved. The SQL Query Engine (SQE) does not have these restrictions. For additional

information regarding SQE, see the Links section of this course.



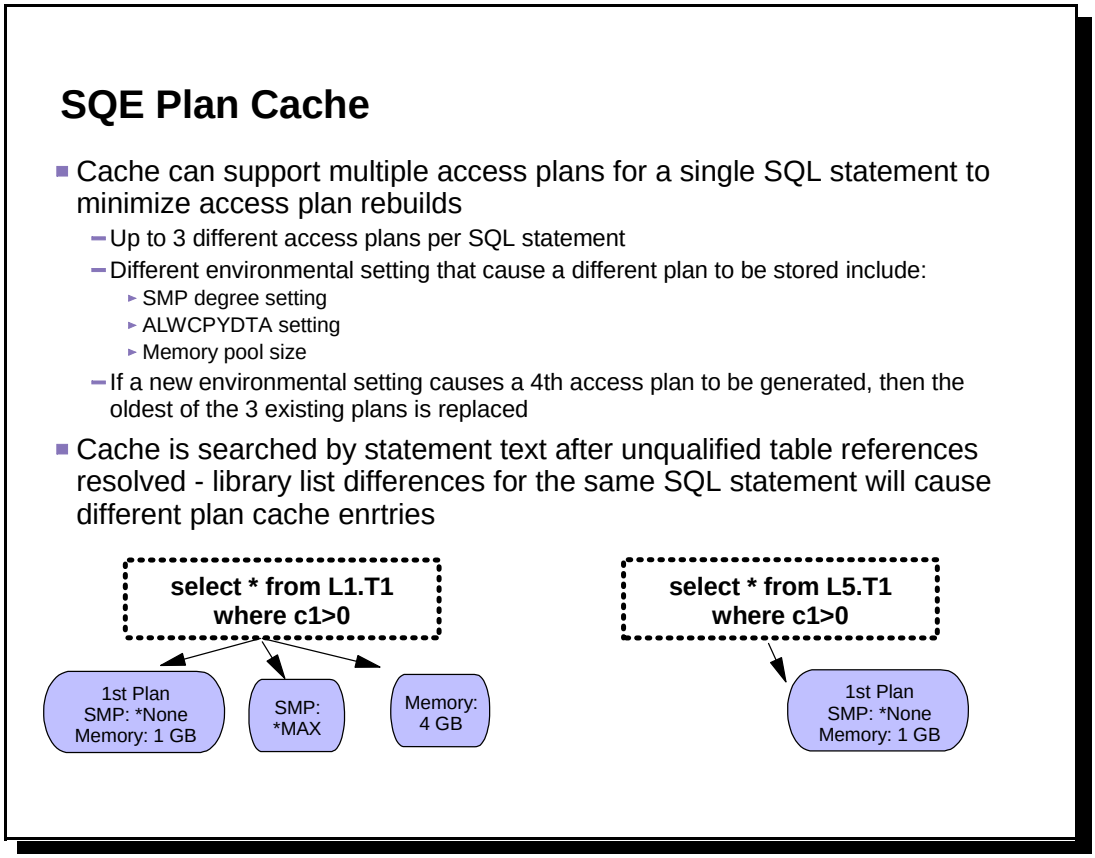
SQE Plan Cache

The SQL Query Engine has more flexibility and efficiency with access plans due to its Plan Cache design. SQE uses a self-managed plan cache to get the highest reuse of access plans associated with the most common SQL requests. Better plan reuse means less access plan builds and rebuilds, usually improving performance. Common SQL statements that are frequently executed by different programs and interfaces can now share the same implementation plan instead of each creating its own copy.

Like CQE, SQE has the ability to update access plans in its cache on demand to utilize any new indexes or statistics that may have been added to the system to improve performance. The advantage of SQE's dynamic updating of plans is that every invoker of that SQL statement will automatically benefit from the updated access plan instead of each invoker updating its own copy of the implementation plan.

The SQE Plan Cache applies to any SQL request it executes on a server, regardless of the interface. Since the SQE Plan Cache is used for SQL requests from any interface, the composition is different in V5R2 for access plans stored in memory, program, and package objects. Prior to V5R2, an access plan was a self-contained object. As this chart shows, access plans generated by SQE are now split in two. The SQE Plan Cache stores the optimized portion (e.g., index scan access method) of the access plan. The access plan components associated with the SQL statement text and other information is left in the original access plan location with a virtual link to the plan in the plan cache.

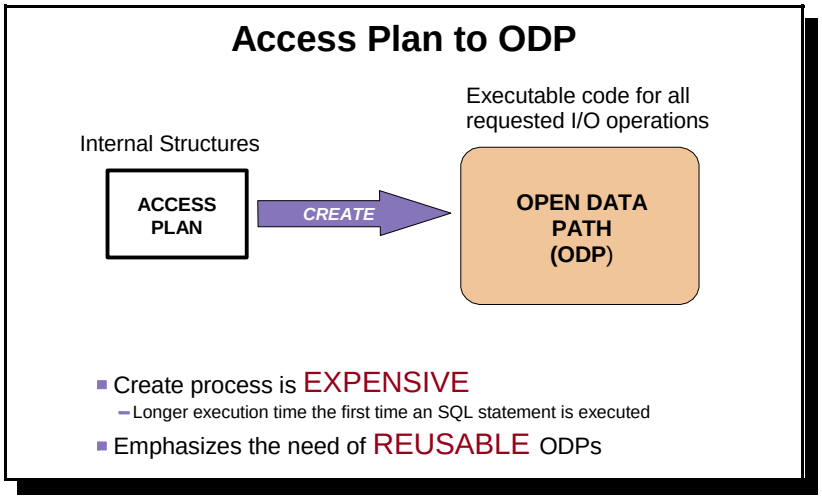
The SQE Plan Cache is self-managed, so DB2 UDB for iSeries automatically allocates space and monitors the usage of this space. As the space limit is approach, DB2 UDB for iSeries automatically removes plans based on the number of times an access plan was used and how long it has been since the plan was used. There are no user tools at this time for viewing and manipulating the contents of the SQE Plan Cache.



SQE Plan Cache

Another unique capability of SQE is the ability to store up to three access plans for a single SQL request. This design again minimizes access plan rebuilds since SQE can switch back and forth between the different plans. A different parallel processing degree for DB2 SMP and a larger memory pool size are among the settings that can cause different plans for a single SQL statement.

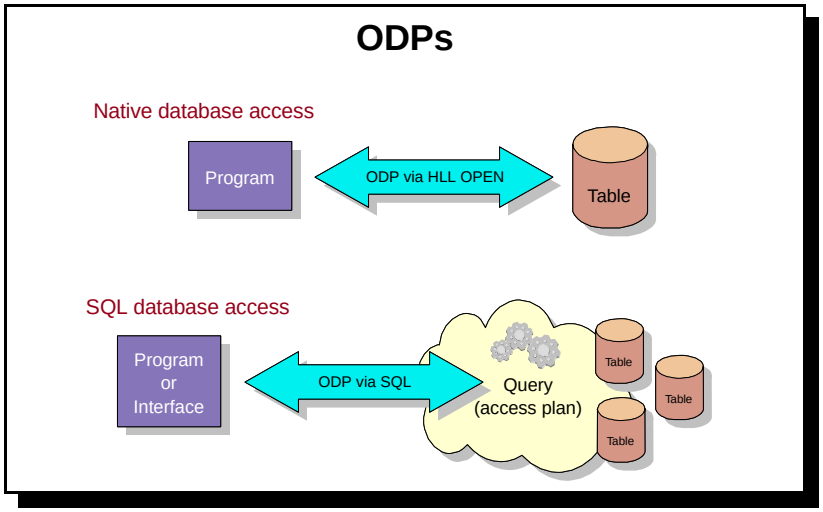
A different library list will not cause multiple plans for an SQL statement. If an SQL request contains unqualified table names, then a plan cache entry is not created for that statement until after the library (or schema name) has been resolved. As this example shows, the statement, SELECT * FROM t1 WHERE c1 > 0, will have a different plan cache entry for each library list that it is executed with.



Access Plan to an Open Data Path (ODP)

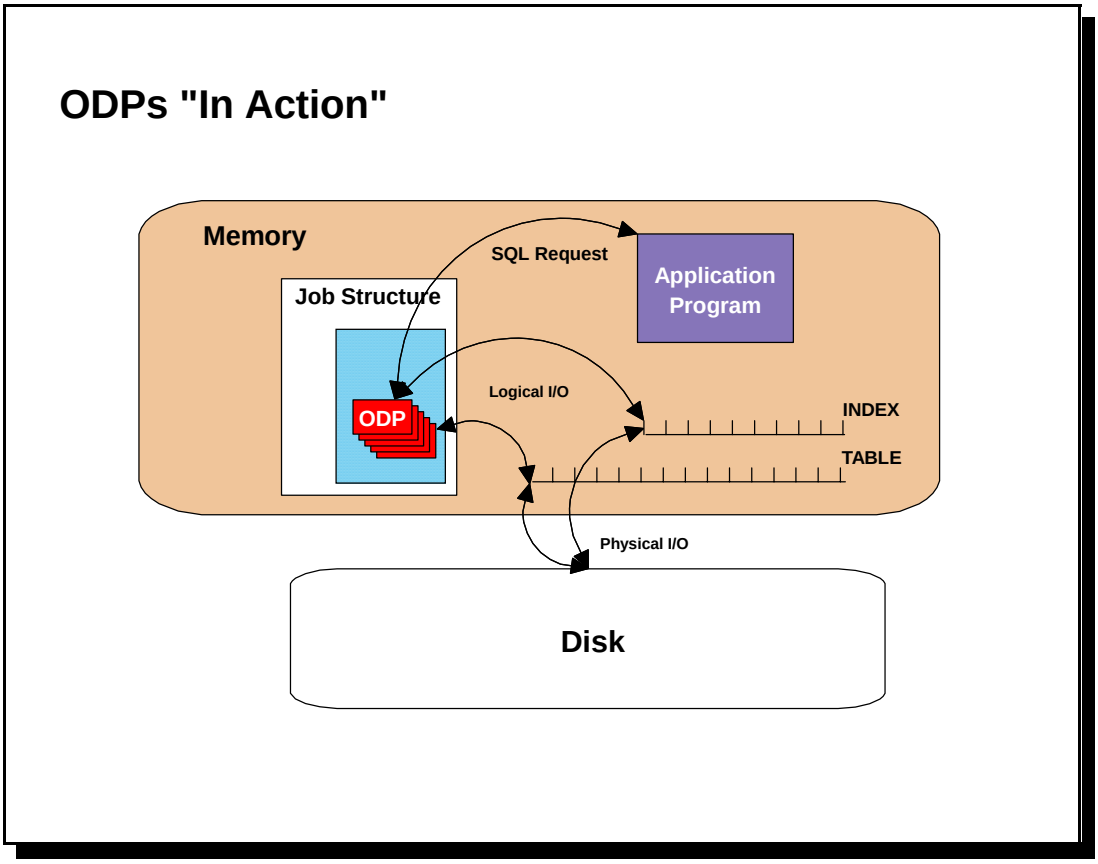
As mentioned earlier, the Open Data Path is the actual pipe for moving data between the database and your applications. The creation of the Open Data Path is very expensive, in terms of performance, on iSeries servers. Avoiding the creation of ODPs (a.k.a., Full Open) is the key to delivering high-performing SQL solutions on iSeries server.

We will review those events or situations where ODPs can and cannot be reused.



ODPs

People experienced with using native iSeries interfaces are familiar with pre-opening files to avoid the cost of a full open. Unfortunately, with SQL, there's no way to "pre-open" a table or SQL statement. Therefore, you must rely on your application design, coding, and the server to avoid full Opens.

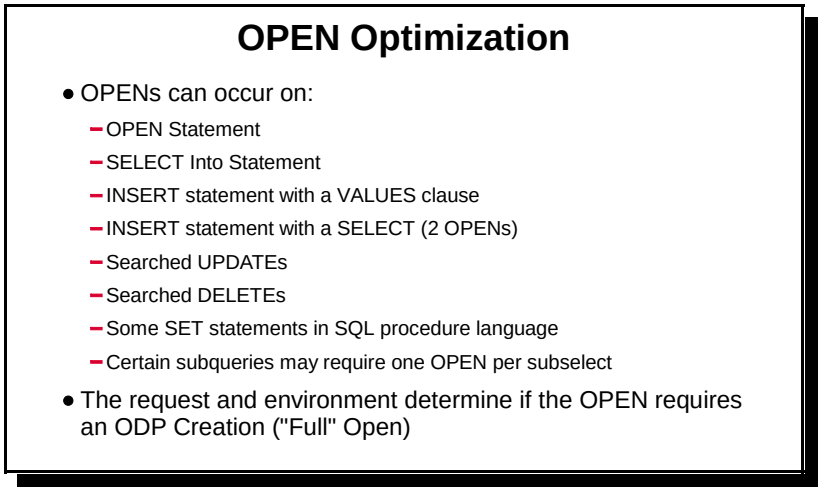


ODPs “in Action”

Many users want to see where ODPs live, and how they are used to implement SQL statements. ODPs live in working memory of the job associated with the SQL request. You can see in this chart that the ODP is the object coordinating and controlling the low-level movement of the data to and from the application.

In general, an ODP is created when the application program opens a file. In the case of SQL, an ODP is created for each unique SQL request in the application. Since ODPs are stored in memory, they have a direct impact on the memory footprint of an application.

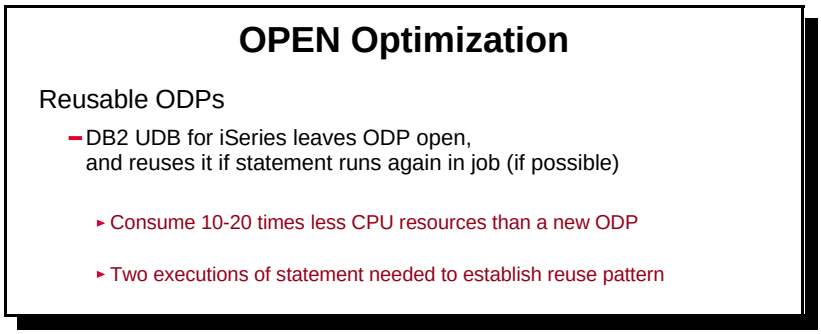
.



OPEN Optimization — OPENs

Review the list of situations shown in the graphic for situations where OPENs can occur. All of these SQL requests use cursors underneath the covers, therefore causing OPENs.

Searched UPDATEs and DELETEs basically cause any UPDATE or DELETE statements to go down the OPEN path. The only exception is positioned UPDATEs or DELETEs done via a declared cursor.

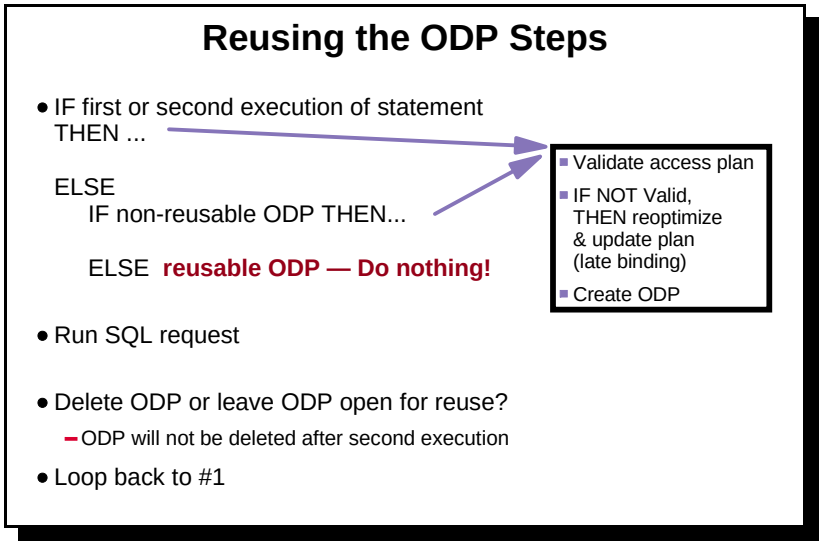


OPEN Optimization — Reusable ODPs

DB2 UDB for iSeries tries to minimize the number of ODP creations ("full opens") within a job by reusing existing ODPs whenever possible. Creating an ODP requires 10 to 20 times more system resources than reusing an existing ODP.

Reuse of ODPs is only of benefit when the same statement is executed multiple times within a job. That's why DB2 UDB for iSeries does not start reusing ODPs until after the second execution of the same statement. (Later, we will cover a data area available for customizing the ODP Reuse behavior).

Be aware that there are some ODPs that cannot be reused, due to their particular implementation.



Reusing the ODP Steps

Here's the logic flow that DB2 UDB for iSeries goes through to determine if the ODP can be left open for reuse:

* If this is the first or second execution of a statement, then validate the access plan. If it is not a valid access plan, then reoptimize and update the plan (that is, do a late binding). Then, create the ODP.

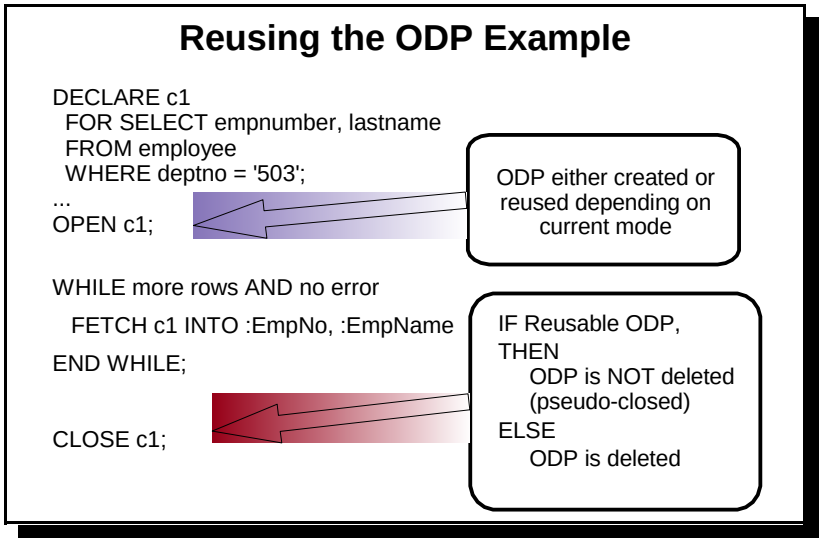
* If this is not the first or second execution of a statement ***AND*** if this is a non-reusable ODP, then validate the access plan. If it is not a valid access plan, then reoptimize and update the plan (that is, do a late binding). Then, create the ODP.

* If this is not the first or second execution of a statement ***AND*** if this is a reusable ODP, then do not create an ODP.

* Run the SQL request.

* Determine whether to delete the ODP or to leave it open for later use. NOTE: the ODP will not be deleted if this is a second execution of the ODP.

* Loop back to the top of this logic flow ...

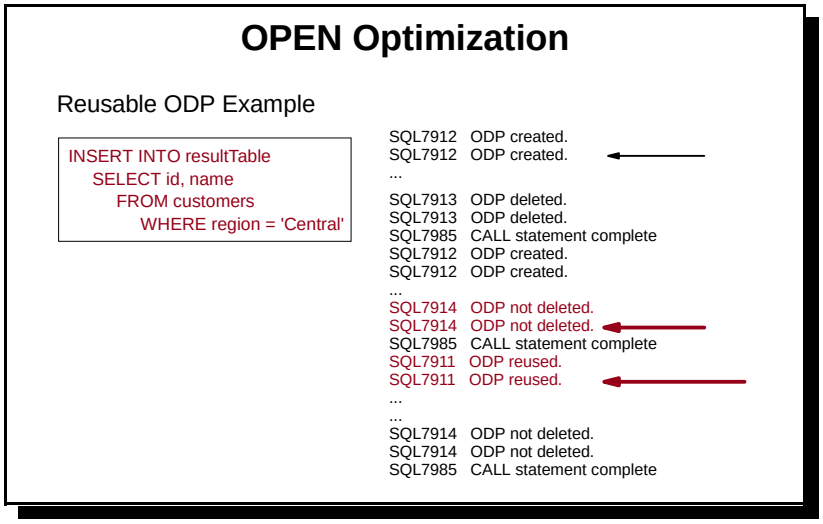


Reusing the ODP Example

Here's an example of how reusable ODPs might work in an SQL application. The OPEN of the cursor is when DB2 UDB for iSeries goes through its logic to determine if an ODP needs to be created or reused. An ODP will have to be created for Cursor C1 the first and second time that the OPEN is executed within the job — and then subsequent executions can benefit from reusable ODPs.

Pseudo-close means that the ODP and cursor are logically closed from an application and SQL perspective, but remain active so a Full Open does not have to be performed when the statement is run again. Internally, this is known as maintaining a ready, pseudo-open.

The pseudo-closed state is similar to the Recycle Bin found on Microsoft® Windows® desktops. The objects placed into the Recycle Bin are logically deleted, but not physically deleted until the Recycle Bin is emptied.



OPEN Optimization — Reusable ODP Example

Here's a look at the feedback that you will get concerning ODP Reuse from the iSeries SQL engine when running your SQL in debug mode. In this example, the Same Stored Procedure is called three different times in the same job to fetch "Central" customers into a temporary result table.

Note that the ODP is not deleted, but is reused on subsequent executions of the stored procedure. The "ODP not deleted" message (Message ID SQL7914) is THE indicator of ODPs being left open for reuse — the "ODP Reused" message is NOT generated every time the ODP is reused. (Different interfaces and different sequences of SQL statements can cause the “ODP Reused” message to not be signaled).

1st Mini Quiz

1. For Static SQL, an Access Plan is created for each SQL statement and:

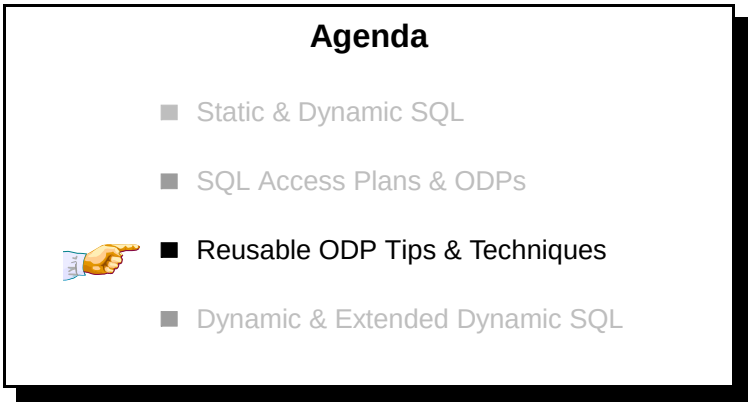
   - ☑ Stored into the program object
   - ❑ Validated by the precompiler
   - ❑ Eventually slows the system if unchecked
   - ❑ Stored in working memory

2. Creating an ODP requires _____ more system resource than reusing an existing ODP.

   - ❑ 50%
   - ☑ 10 to 20 times
   - ❑ 25%
   - ❑ 50 times

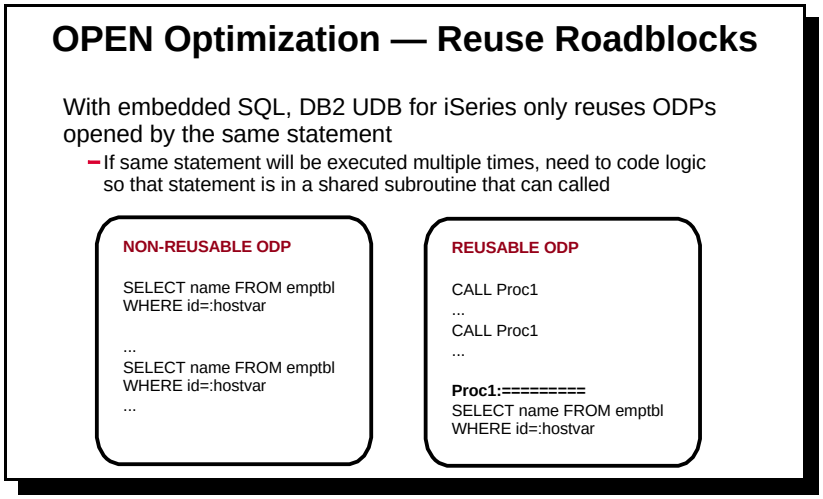
3. DB2 UDB for iSeries attempts ODP Reuse when:

   - ❑ Encountering an SQL REUSE statement in the application
   - ❑ Specifying the precompiler
   - ❑ The same SQL statement has been executed more than twice on the system
   - ☑ The same SQL statement has been executed more than twice within a job or connection



Reusable ODP Tips and Techniques

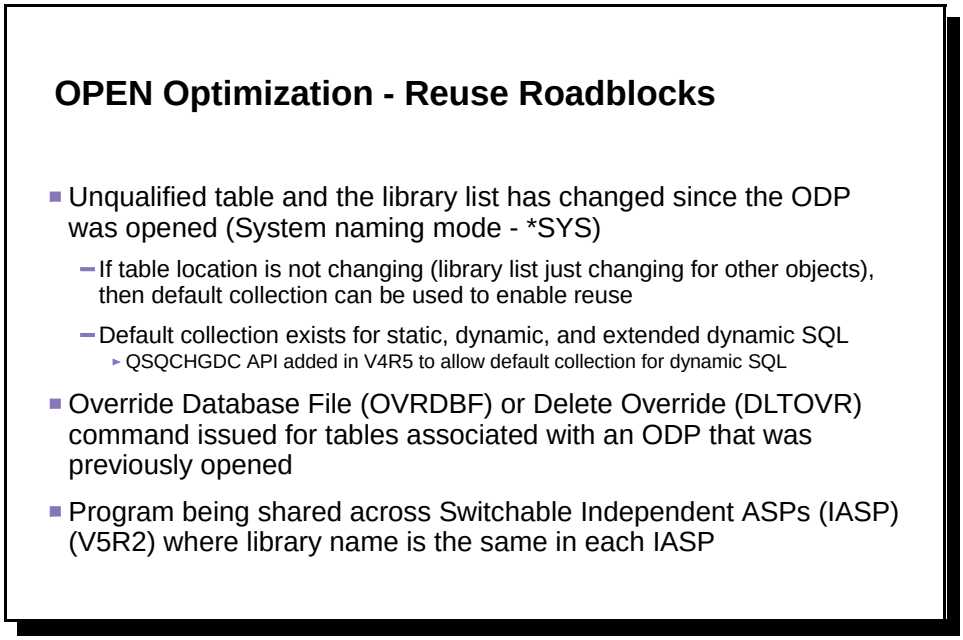
Now that you understand the benefits gleaned by DB2 UDB for iSeries leaving ODPs open for reuse, this next section will cover the designs and requests that can prevent DB2 UDB for iSeries from reusing existing ODBC Open Data Paths.



OPEN Optimization — Reuse Roadblocks

DB2 UDB for iSeries can only reuse ODPs for a single, embedded SQL statement. Different instances of the same SQL statement cannot share the same ODP. Thus, if the same statement will be executed multiple times within a program call, you should code the logic so that the statement is in a shared subroutine that can be called. Then, DB2 UDB for iSeries is able to reuse the ODP. See the example shown in the graphic.

NOTE: Dynamic SQL is discussed in the following section. Almost all of these open and reuse tips and restrictions apply to all SQL interfaces, unless explicitly noted.



OPEN Optimization — Reuse Roadblocks (continued)

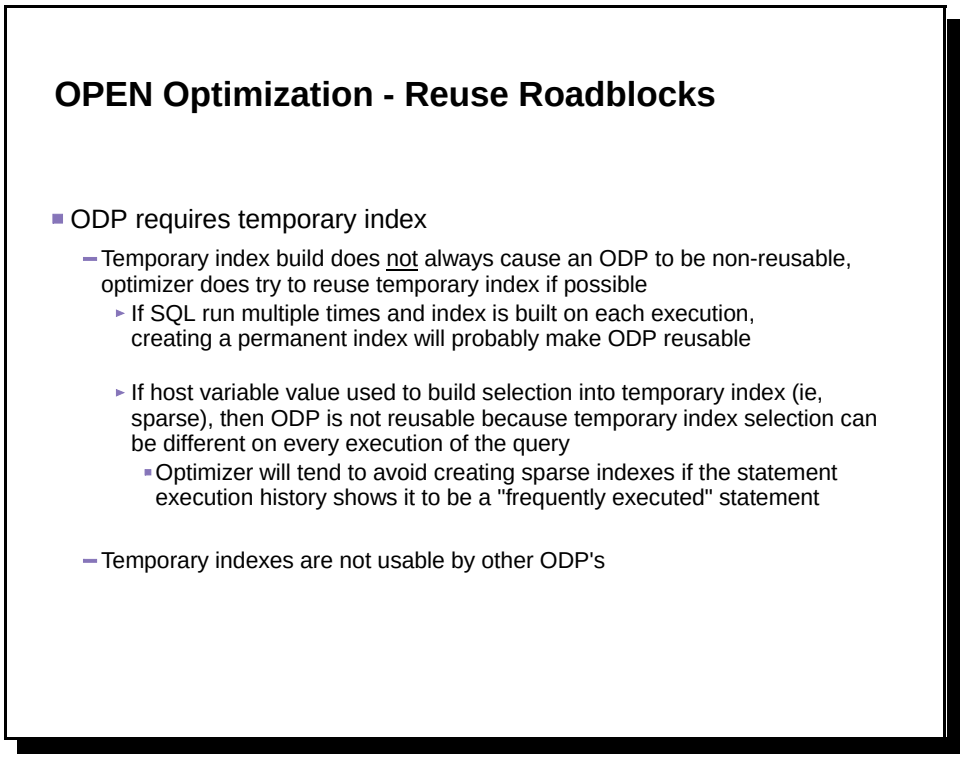
If unqualified SQL tables reside in your SQL requests (that is, you have not coded them in the library/collection/schema name) and you are running with system naming mode (*SYS) so that the system determines the location of the table, then changing the library list will cause an ODP creation. If the table location has changed, the existing ODP cannot be reused because it is pointing at the wrong version of a wrong table.

Here's an example of how that might happen. An international company experienced performance problems when using a single SQL program to run against different company databases that resided in different libraries. Every time the library list changed for a different country, the opened ODP was useless and had to be deleted.

If the database table location is not really changing (the library list might be changing to just switch the location of data queues, data areas, etc.), then you should use a precompiler option that specifies a default collection (library) for unqualified tables. With the default collection, the database does not have to search for the location of the table.

SQL naming mode (*SQL) is not impacted by this, because it does not search the library list for unqualified tables. It only searches the library with the same name as the user's ID running the job.

The system override commands [Override Database File (OVRDBF) and Delete Override (DLTOVR)] can point to different versions of the same table. In those cases, the ODP must also be recreated.

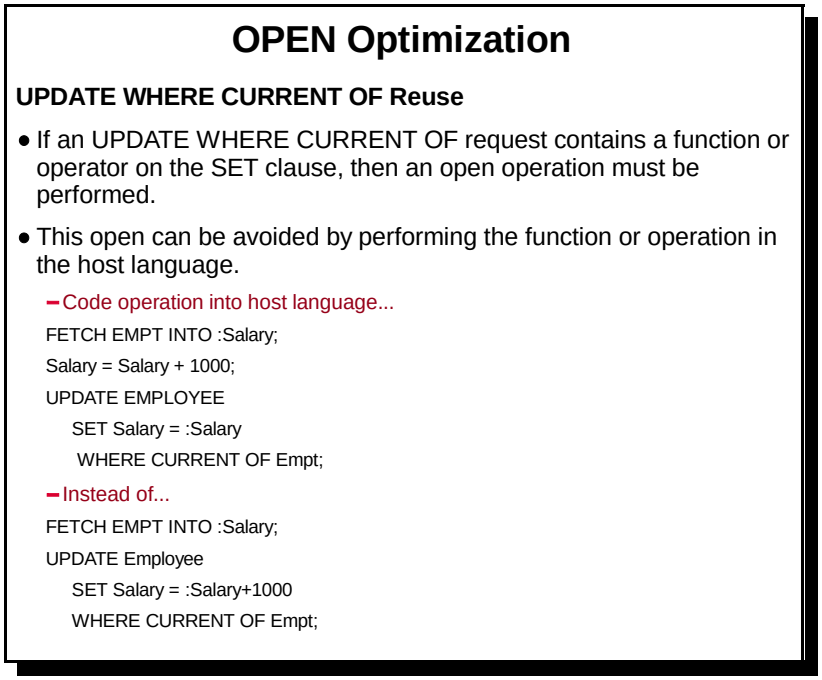


OPEN Optimization — Reuse Roadblocks (continued)

A temporary index does not always make the ODP non-reusable. The only real indicator that the temporary index is allowing the associated ODP to be reusable is if a temporary index build appears during Full Open and then the SQL statement goes into reusable mode.

If a sparse (select/omit) temporary index is created to build host variable selection into the index, then the ODP is not reusable because the index selection can potentially be different on every execution of the query.

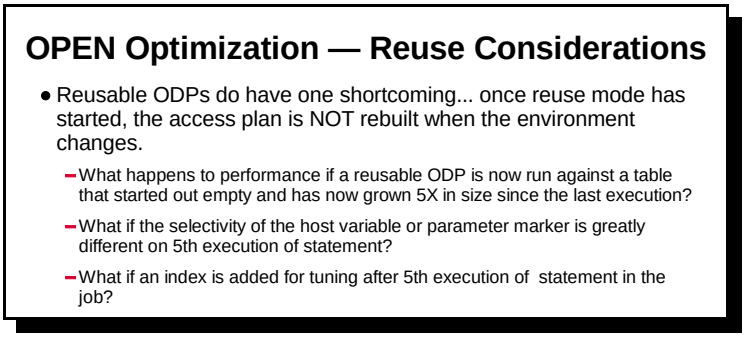
Temporary indexes can never be shared by other ODPs.



OPEN Optimization — UPDATE WHERE CURRENT OF Reuse

Normally, an UPDATE WHERE CURRENT OF request does not require an open operation, that is, the creation of an ODP. However, if the SET clause includes a function or operator, then a full open must be performed.

To avoid this, you can try changing the request to have the high level language perform the requested function or operation, and then use the computed result on the UPDATE request. (See the code sample shown in the graphic.)



OPEN Optimization — Reuse Considerations

You should be aware that reusable ODPs have one major drawback. When ODPs are in reusable mode, DB2 UDB for iSeries is unable to react to environment changes, such as table size increases, new indexing, etc., and cannot rebuild the associated access plan.

Reusable ODPs cause DB2 UDB for iSeries to acquire shared locks on the associated database objects, and columns and indexes cannot be deleted once they are in reusable ODP mode.

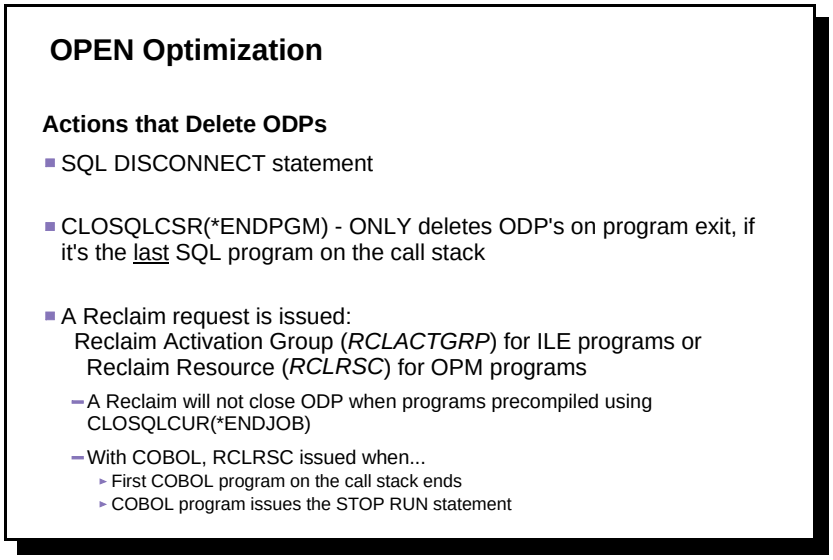
Therefore, you will need to consider the following questions:

* What happens to performance if a reusable ODP is now run against a table that started out empty and has grown five times in size since the last execution?

* What if the selectivity of the host variable or parameter marker is greatly different on the fifth execution of the statement?

* What if an index is added for tuning after the fifth execution of a statement within the job?

These are cases that you, as a programmer or administrator, have to be aware of when designing your application. On the next couple of charts, we cover some of the options you have available for forcing the deletion of an ODP to avoid these problems.



OPEN Optimization — Actions that Delete ODPs

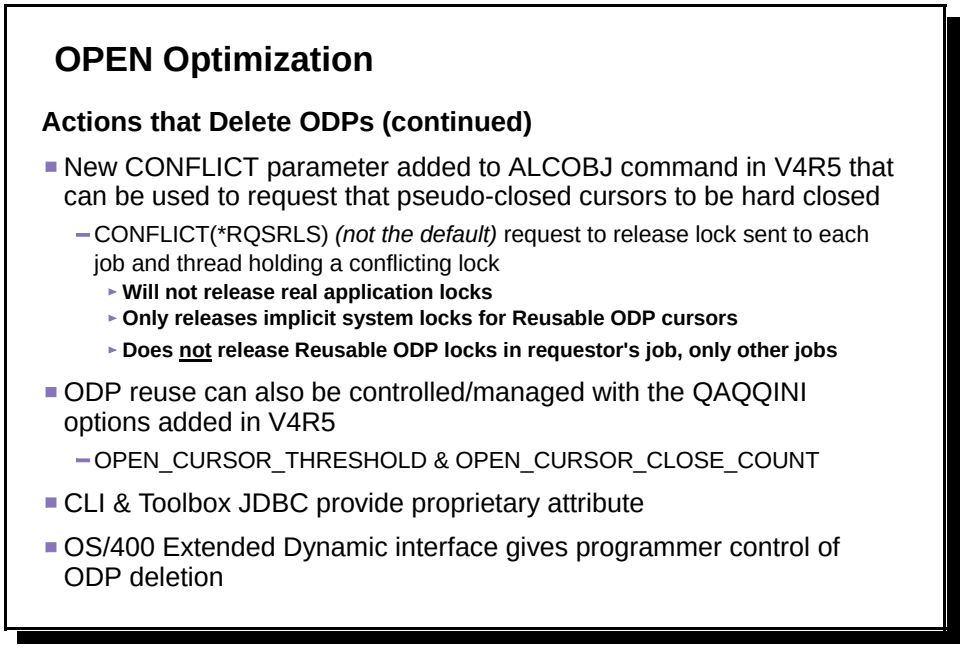
We just covered some issues where ODP reuse can have a negative effect, instead of a positive effect. This graphic and the next chart presents the actions that force the ODP to be deleted. Due to the performance benefits of ODP reuse, these techniques should only be used in a limited fashion.

The SQL DISCONNECT statement forces the ODP to be deleted! SQL DISCONNECT is typically only used with a remote DRDA connection; though it could be used as a local database access technique to prevent ODP reuse from occurring. Use this technique carefully, because it would delete the ODP for all SQL statements in a connection. You would likely use the technique with a single SQL statement in a program by itself.

It may appear that the Close SQL Cursor precompiler option allows you to force an ODP to be deleted when a program call completes with the End Program (*ENDPGM) value. But ODPs will only be deleted with the End Program value when the program being ended is the last one on the call stack containing SQL requests. If the program is not the last program on the call stack performing SQL, then the End Program setting is ignored.

When a Reclaim request is issued, it forces the ODP to be deleted! Specifically, Reclaim Activation Group (RCLACTGRP) for ILE programs or Reclaim Resource (RCLRSC) for OPM programs cause the ODP to be deleted. However, a Reclaim will not close ODP when programs are precompiled using the End of Job value for the Close SQL Cursor parameter (CLOSQLCUR (*ENDJOB)).

NOTE: For those working with COBOL, be aware that the Reclaim Resource (RCLRSC) request is automatically issued when the first COBOL program on the call stack ends or when the COBOL program issues the STOP RUN statement.



OPEN Optimization

The Allocate Object (ALCOBJ) command can be used to delete the ODPs for a specific DB2 UDB for iSeries table. If you have the scenario where a work table that was empty at the beginning of a batch process is causing performance problems because it now contains thousands of rows, the Request Release value (*RQSRLS) on the CONFLICT parameter can be used to force the deletion of only ODPs associated with that work table. Once the ODP is deleted, the query optimizer will be invoked on the next execution and be able to choose a different access method now that it sees how large the work table has gotten. The options discussed on the earlier chart are not very granular because they cause all the ODPs associated with a connection or job to be deleted.

By default, DB2 UDB for iSeries keeps 512 ODPs active within a job or connection. If this limit is exceeded, than DB2 UDB for iSeries deletes the least-recently used ODPs. To force ODPs to be deleted more often, the number of active ODPs limit can be set to a value smaller than the default by setting the QAQQINI OPEN_CURSOR_THRESHOLD parameter. The OPEN_CURSOR_COUNT parameter controls how many ODPs are deleted once the threshold or ODP limit is hit.

One of the benefits of using the proprietary SQL Process Extended Dynamic QSQPRCED API interface is that the programmer can control the reuse and delete of ODPs. The SQL Call Level Interface (CLI) and Toolbox JDBC drivers also offer proprietary statement attributes to force an ODP to be created each time that an SQL statement is executed.

2nd Mini Quiz

1. ODPs can be shared and reused by different jobs.

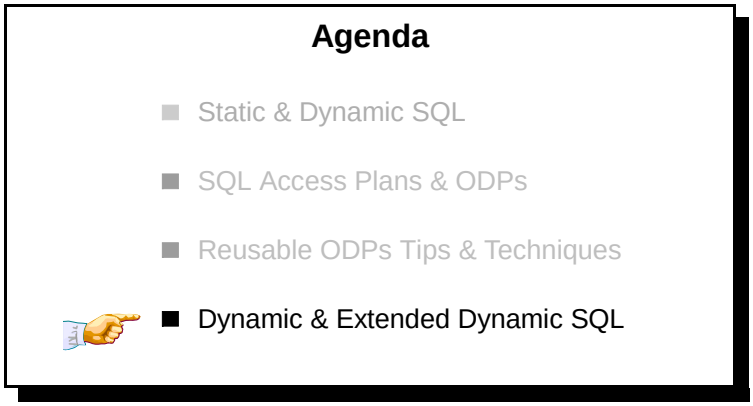
   - ❑ True
   - ☑ False

2. Temporary indexes always cause the associated ODP to be non-reusable.

   - ❑ True
   - ☑ False

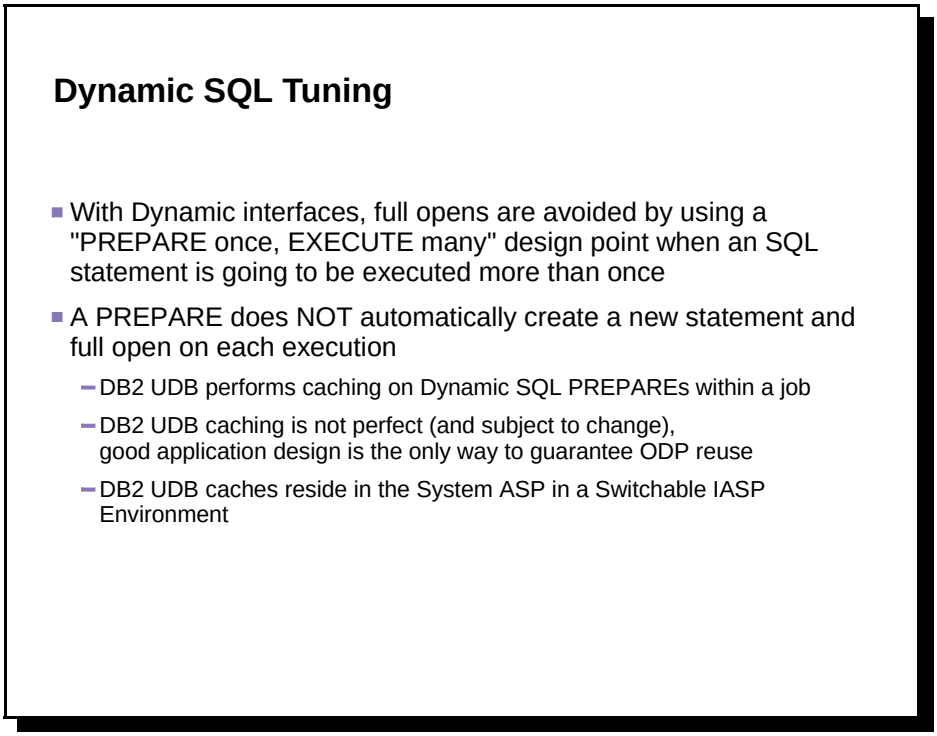
3. Reusable ODPs have one shortcoming:

   - ❑ They may wear out after multiple IPLs
   - ❑ They cause additional memory to be used on each reuse
   - ☑ In reusable mode, the DB2 UDB for iSeries query optimize cannot react to environment changes (e.g., creation of indexes)
   - ❑ Access Plans are constantly rebuilt



Dynamic and Extended Dynamic SQL

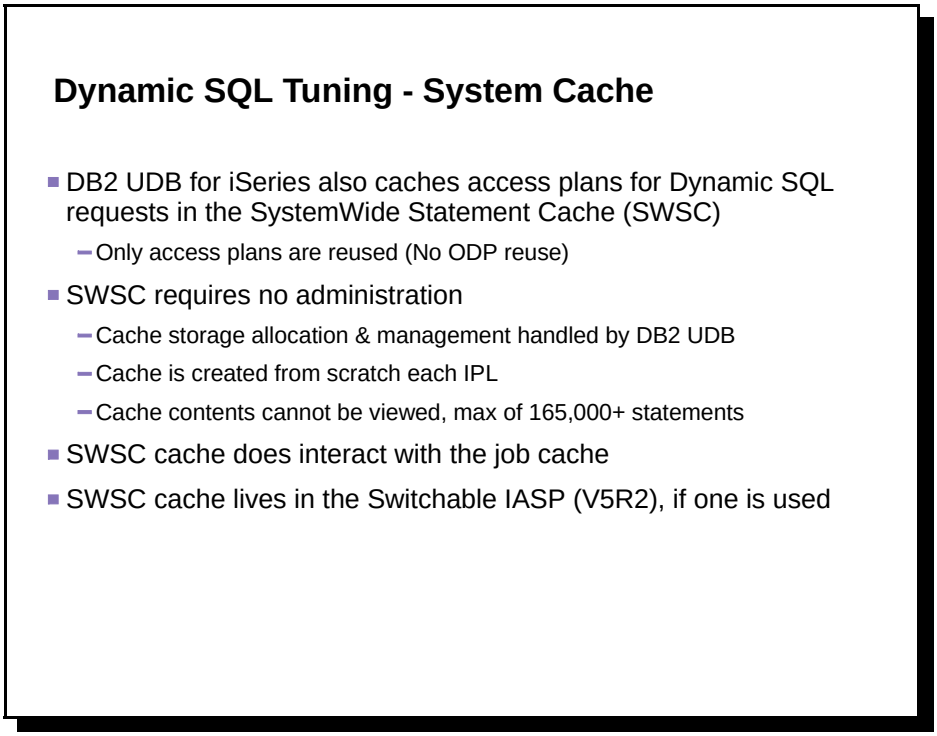
We have covered basic SQL performance constructs and issues. Now, let's look at the nuances of Dynamic SQL on DB2 UDB for iSeries.



Dynamic SQL Tuning

Here's a good rule to follow: "PREPARE once, EXECUTE many." This follows the idea of avoiding full opens and reusing ODPs. This is a key design point that often has to be changed in SQL applications that are ported over to DB2 UDB for iSeries. Preparing a frequently executed statement once is also more efficient for other database products. DB2 UDB for iSeries, however, has a bigger performance penalty than other database products when the same statement is prepared multiple times within a connection.

To minimize the number of Full Opens, DB2 UDB for iSeries tries to automatically cache the ODPs for Dynamic SQL requests at the connection (job) level. Thus, an SQL Prepare does not always cause the creation of an ODP. No caching algorithms are perfect, thus the only way to guarantee ODP reuse is to use the design technique of preparing a statement once and executing it many times.

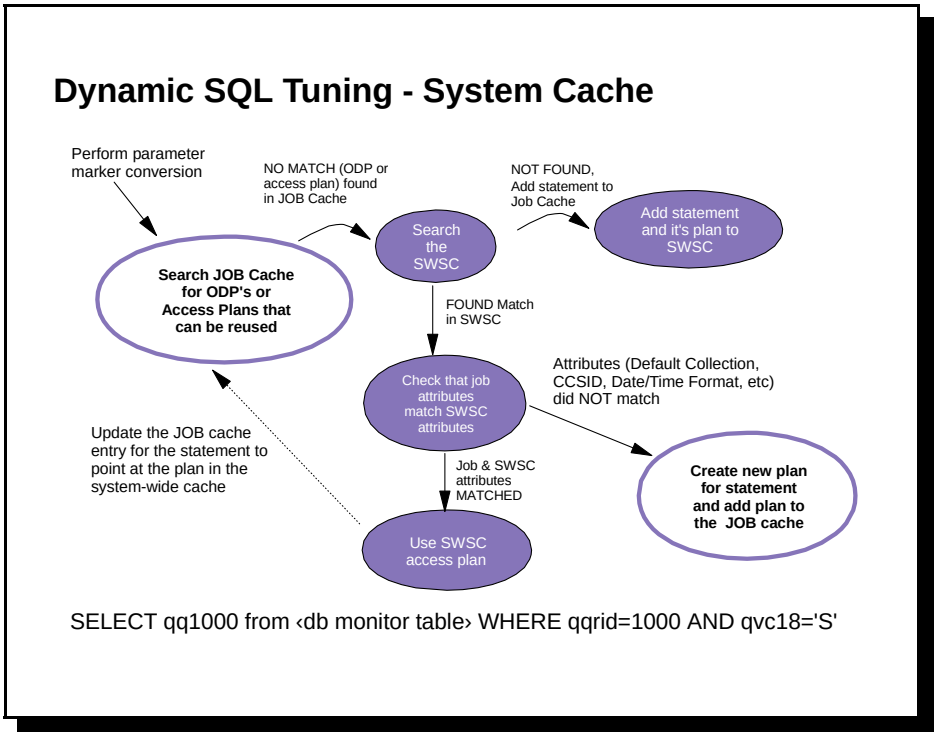


Dynamic SQL Tuning — System Cache

Let's examine the SystemWide Statement Cache (SWSC) that DB2 UDB for iSeries uses for dynamic SQL tuning. First, be aware that with SWSC, only access plans are reused — there is no ODP reuse. The SWSC does not require administration. Instead, cache storage allocation and management are handled by DB2 UDB for iSeries. The SWSC is created from scratch during each IPL.

Cache churn and contention is avoided by DB2 UDB for iSeries by only allowing limited access plan updates. In these cases, the query optimizer will build a temporary access plan instead of using the cached access plan. Therefore, you might think about IPLing (restarting) the system after the database has been tuned for your application.

The SQE Plan Cache discussed on an upcoming chart solves the problem of limited updates to cached access plans.



Dynamic SQL Tuning — System Cache (continued)

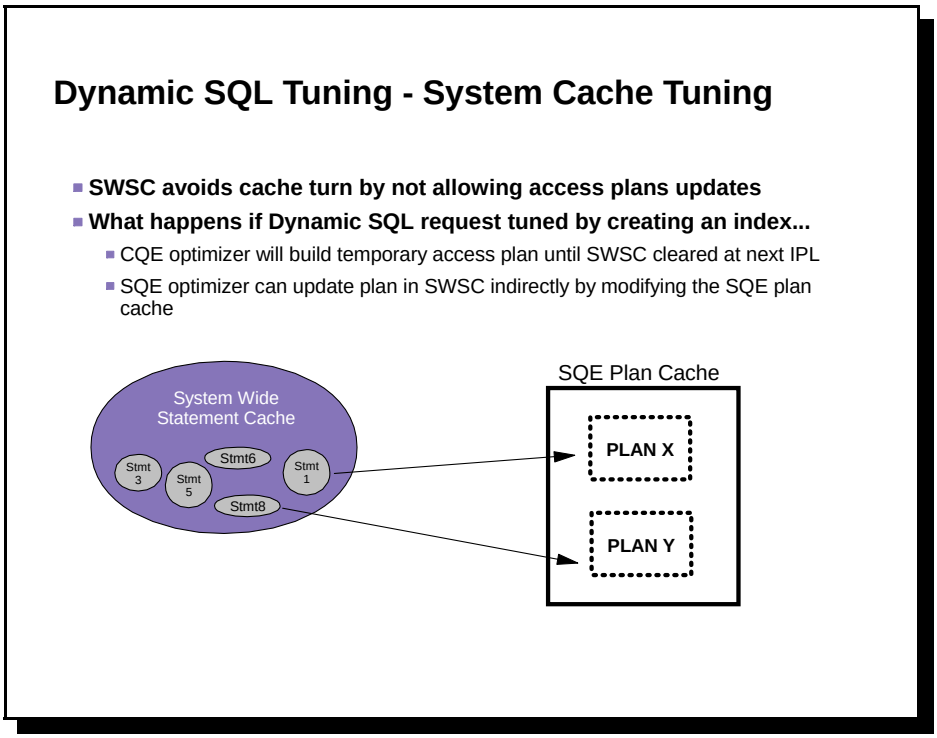
This graphic is a representation of the job and system cache processing steps that DB2 UDB for iSeries exercises on each Dynamic SQL request.

First, the job (connection) cache is searched to determine if a cached access plan or ODP can be used for the requested SQL statement. If a cached plan or ODP cannot be found, then DB2 UDB for iSeries will search the SWSC for a cached access plan.

If DB2 UDB for iSeries finds a match in the SWSC, then it next checks to make sure the job attributes (CCSID, date format, etc.) match those of the access plan stored in the SWSC. If the job attributes are in sync with the SWSC access plan, then the access plan cached in the SWSC will be used. In addition, the job cache will also add a new entry to point at the SWSC access plan — the next time this Dynamic SQL statement is run in the job, only the smaller job cache will have to be searched.

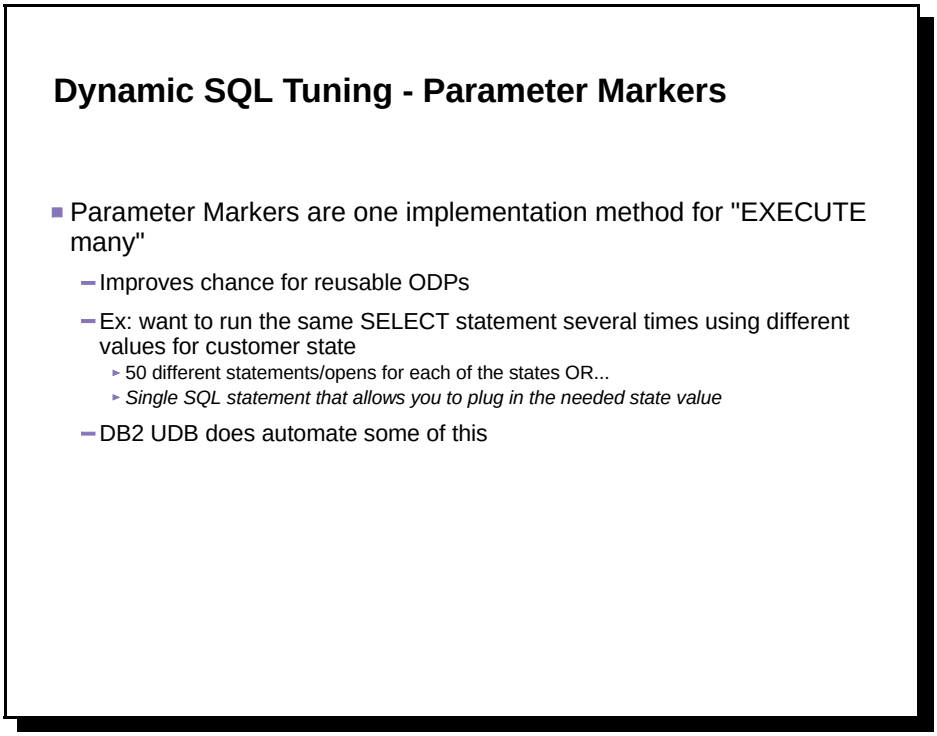
If no match is found in the SWSC, then a new access plan is built and placed into the SWSC. An entry is also placed in the job cache to point at this new plan in the SWSC.

Take a few minutes to study this chart before continuing with this course.



Dynamic SQL Tuning - System Cache Tuning

As you learned earlier, the SQE Plan Cache stores the optimized portion (e.g., index scan access method) of the access plan. This means that the SQE Plan cache provides a solution to allow multiple updates to cached access plans in the System-Wide Statement Cache. When the SQE optimizer updates an access plan, it only has to change the SQE Plan Cache entry. This allows the SQE optimizer to update the SWSC indirectly as many time as it wants. The rest of the cached access plan in the SWSC does not need to be updated since, for SQE, it only contains the SQL statement text and other environment information. Thus, if your application primarily uses SQE, there is no need to restart (IPL) the system to clear the SWSC after your application and database have been tuned.



Dynamic SQL Tuning — Parameter Markers

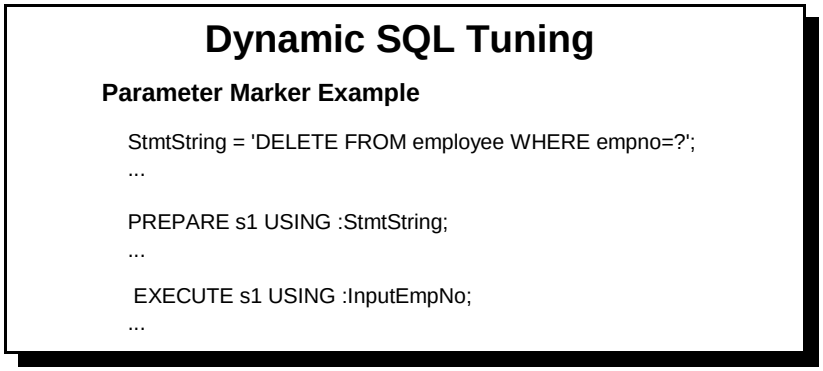
Parameter markers are an effective way of making dynamic SQL requests and their associated ODPs easier to reuse.

Although DB2 UDB for iSeries automatically transforms some dynamic SQL requests to use parameter markers, application designers shouldn't rely on this system support. The favored approach is to utilize parameter markers, which is one implementation method for “EXECUTE many”.

Parameter markers offer the advantage of improving the chance that an ODP can be reused.

For example, let’s say that you want to run the same SELECT statement many times, but you want the option to pick different values for “customer state.” You could write 50 different statements/opens for each of the 50 possible U.S. states, or you could write a single SQL statement that allows you to plug in the needed state value whenever appropriate.

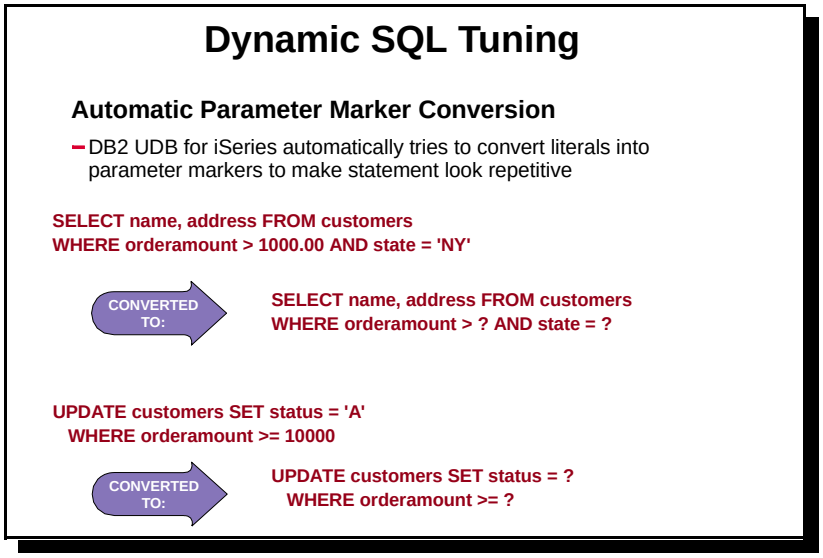
It’s nice to know that DB2 UDB for iSeries automates some of this for you, so the use of parameter markers is easier than you might think.



Dynamic SQL Tuning — Parameter Marker Example

Take a look at the graphic. It is the same dynamic SQL as was shown two charts previously, but now it is utilizing parameter markers, then passing the parameter marker values on the EXECUTE.

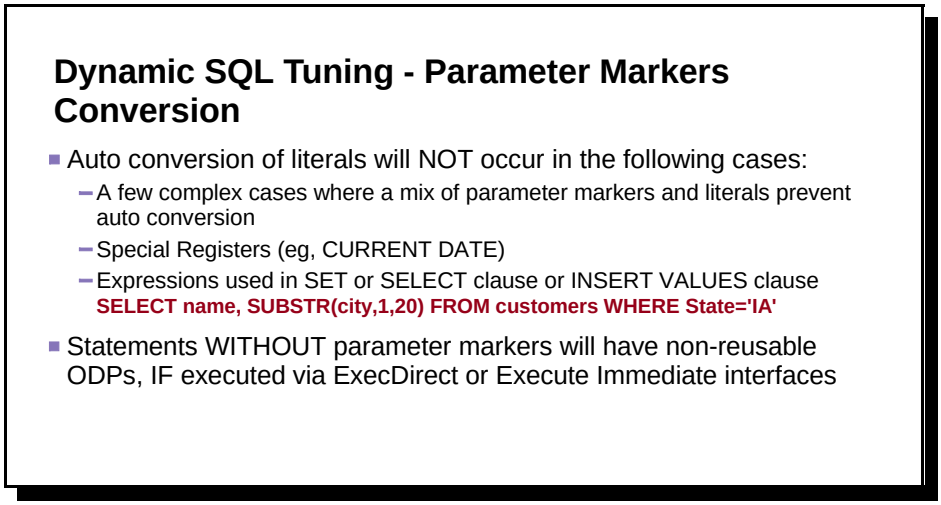
NOTE: On some other SQL-based interfaces, the parameter marker value is assigned by a bind process.



Dynamic SQL Tuning — Automatic Parameter Marker Conversion

You will recall that DB2 UDB for iSeries converts some dynamic SQL request literals into parameter markers in order to make the statement look repetitive. This conversion makes reusing ODPs easier. This graphic shows one example of parameter marker conversion. Look at it briefly to see the logic employed by DB2 UDB for iSeries.

These conversions happen on most dynamic SQL interfaces. The exception is the iSeries Access ODBC and Toolbox JDBC driver — a feature that had to be disabled, because it was negatively affecting client applications that were using parameter markers and extended dynamic packages.



Dynamic SQL Tuning — Parameter Marker Conversion

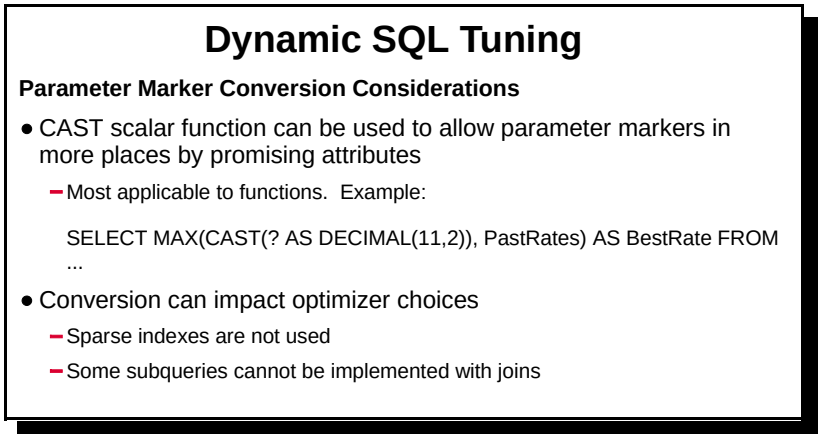
There are certain Dynamic SQL requests that prevent DB2 UDB for iSeries from performing this parameter marker conversion, and examples of each restriction are depicted in this graphic.

* Generally, complex cases that involve a mixture of parameter markers and literals will prevent automatic conversion.

* When SQL statements use special registers, such as CURRENT DATE, CURRENT TIME, etc.

* Also, expressions that are used in the SET or SELECT clause will also prevent automatic conversion.

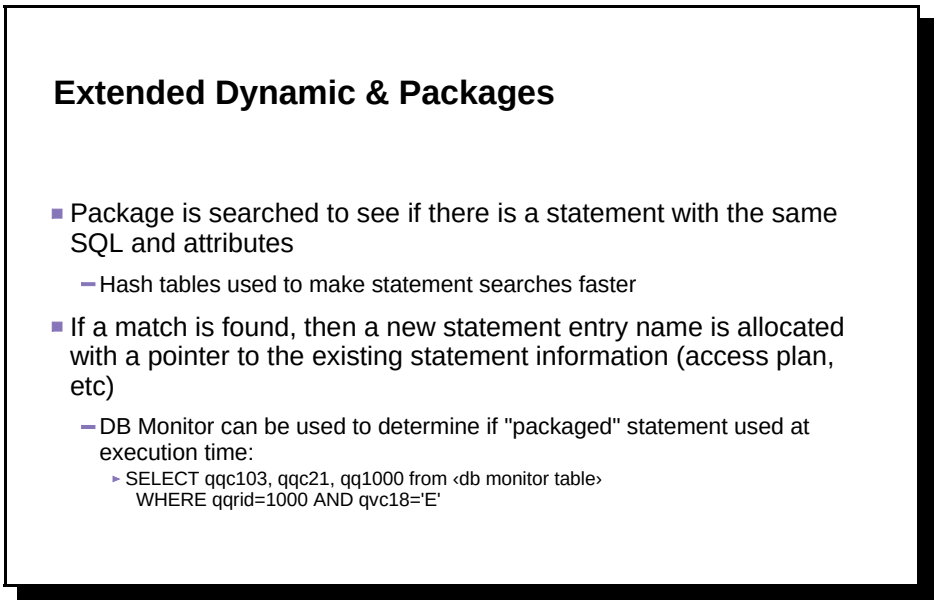
If the SQL statement was executed without parameter markers via an interface such as Execute Immediate or Execute Direct, DB2 UDB for iSeries will always perform a full open since these types of Dynamic SQL interfaces should only be used for statements that are going to be run once within the connection. If the application is using these types of interfaces for statements run many times within a connection, then there's a pretty good chance that the application performance will be subpar.



Dynamic SQL Tuning — Parameter Marker Conversion Considerations

The CAST scalar function allows parameter markers to be used in more places, since it can be used to promise the column/field attributes. Without CAST, there are very few SQL functions that can be invoked with parameter markers.

Parameter markers do limit optimizer choices for implementation. Specifically, sparse indexes are not used, and some subqueries cannot be rewritten as a join.

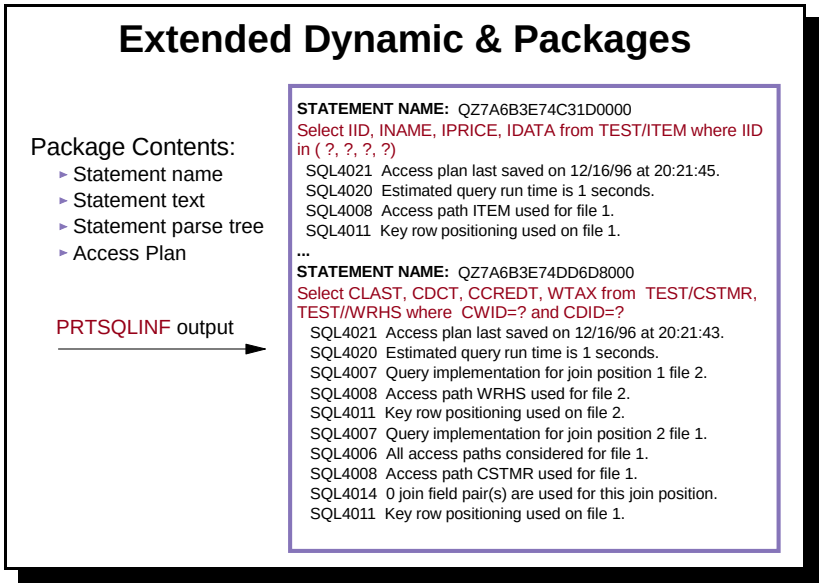


Extended Dynamic and Packages

Extended dynamic improves performance by permanently caching dynamic SQL requests within an SQL package. This almost mirrors how the access plan for embedded, static SQL requests are stored in the program object.

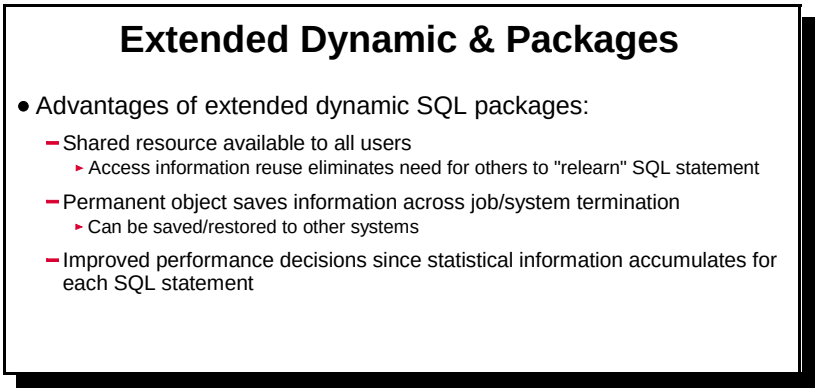
A search of the package is done by statement text, similar to the search that's done for the caching that occurs at the job level for pure dynamic SQL. A hashing algorithm is used to make this statement search very efficient. If a match is found, then a new statement entry name is allocated with a pointer to the existing statement information, including its access plan, etc.

NOTE: Database Monitor data can be analyze to determine if the "packaged" statement was used at execution time. See the sample code shown at the bottom of this chart for details.



Extended Dynamic and Packages — Package Contents

The PRINT SQL INFORMATION (PRTSQLINF) CL command or iSeries Navigator Explain SQL function can be used to display the SQL statements and their access plans stored in a package (or program). This graphic represents the output of that command. Notice that the main package contents for a statement includes the name, text, statement parse tree, and access plan.

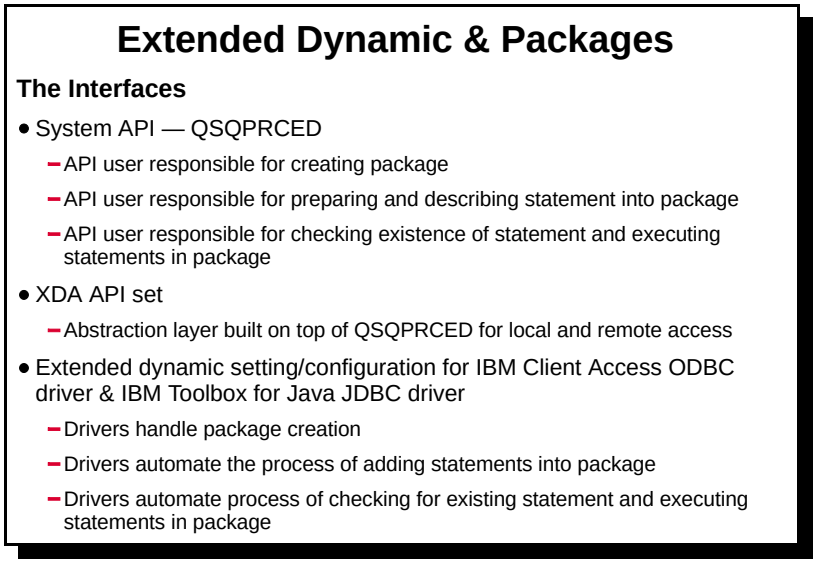


Extended Dynamic and Packages — Advantages of using Extended Dynamic SQL Packages

Packages can share statement access plan information to all system users, in a manner that is similar to embedded SQL access plans that are stored in program objects. This eliminates the need for others to “relearn” the SQL statement.

The package is treated as a permanent object that continues to exist even after job and system termination, just as a program object remains after a job is terminated. Because of its permanency, the package can even be saved and restored to other systems. In addition, because basic statistics are maintained for each statement in the package, the optimizer can make improved performance decisions.

The statistics kept in the object include: number of times the package is used, number of times a packaged statement is executed, and number of rows fetched by a statement. With these pieces of information, statements in an SQL package tend to go into reusable ODP mode after the first execution, which rapidly improves performance — automatically!



Extended Dynamic and Packages — Interfaces

There are three interfaces into iSeries extended dynamic support:

1. A system API called SQL Process Extended Dynamic (QSQPRCED)

2. The XDA API set

3. An option on the IBM-provided ODBC driver and the IBM Toolbox for Java™ JDBC driver.

The system API (QSQPRCED) requires the user to build, manage, and populate the package. That is, the API user must be responsible for: creating the package, preparing and describing the statements into the package, checking for the existence of the statements, and executing the statements into the package.

The XDA API set tries to reduce the complexity and burden from the QSQPRCED API by providing an abstraction layer to be built on top of QSQPRCED for local and remote access.

The ODBC and JDBC drivers allow you to use, and benefit from, extended dynamic support by specifying that option with a mouse click or a single line of code. The drivers handle package creation and automate the process of adding statements into the package. Additionally, the drivers automate the process of checking for existing statements and for executing statements in the package.

3rd Mini Quiz

1. _____ are an effective way of making dynamic SQL requests, and the associated ODPs, easier to reuse.

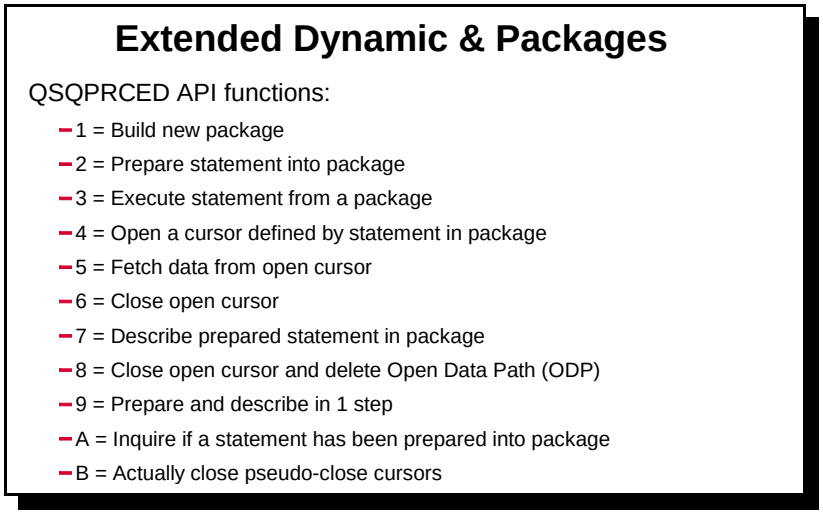
   - ❑ SQL CONNECTs
   - ❑ Automatic PREPAREs
   - ☑ Parameter markers
   - ❑ SQL Optimizers

2. An application that relies on the DB2 UDB for iSeries caching of ODPs is guaranteed to get the same performance as an application using the "Prepare Once, Execute Many" programming technique.

   - ❑ True
   - ☑ False

3. What is an advantage of using Extended Dynamic SQL Packages:

   - ❑ The package can share statement access plan information to all systems users.
   - ❑ The package is permanent across job and system termination.
   - ❑ Improved performance decisions since statistical information is accumulated for each SQL statement.
   - ☑ All of the above



Extended Dynamic and Packages — QSQPRCED API Functions

Look carefully at this graphic. You will see the different QSQPRCED API functions outlined. Programmers can readily understand what's required for using this API to enable extended dynamic SQL access.

Extended Dynamic & Packages

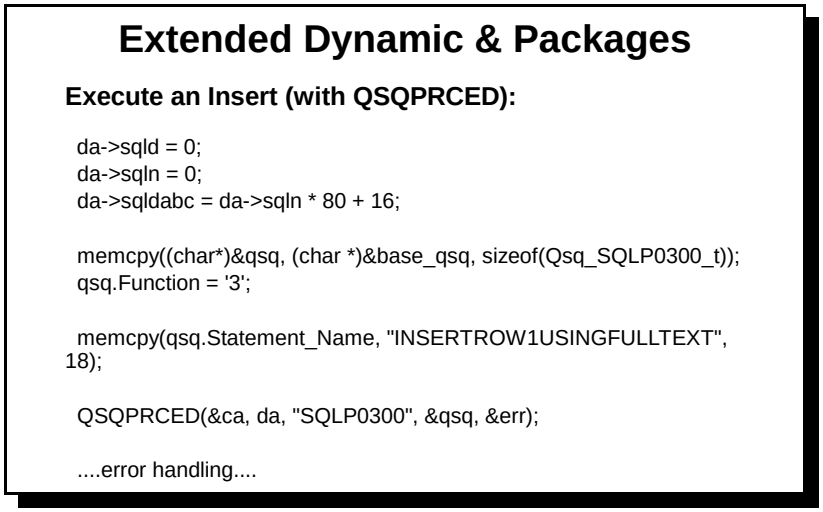
Prepare an Insert (with QSQPRCED):

```
da->sqld = 0;
da->sqln = 0;
da->sqldabc = da->sqln * 80 + 16;

memcpy((char*)&qsq, (char *)&base_qsq,
sizeof(Qsq_SQLP0300_t));
qsq.Function = '2';
qsq.Open_Options = 0x90;        /*  write authority  */

memcpy(qsq.Statement_Name,
"INSERTROW1USINGFULLTEXT", 18);
strcpy(temp, "INSERT INTO ");
strcat(temp, actfile);
strcat(temp, " VALUES (9, 4.5, 2, 'inserted row')");
qsq.Statement_Length = strlen(temp);

memcpy(&qsq.Statement_Length  + 1, temp, strlen(temp));

QSQPRCED(&ca, da, "SQLP0300", &qsq, &err);
```

Extended Dynamic and Packages — Prepare an Insert with QSQPRCED API

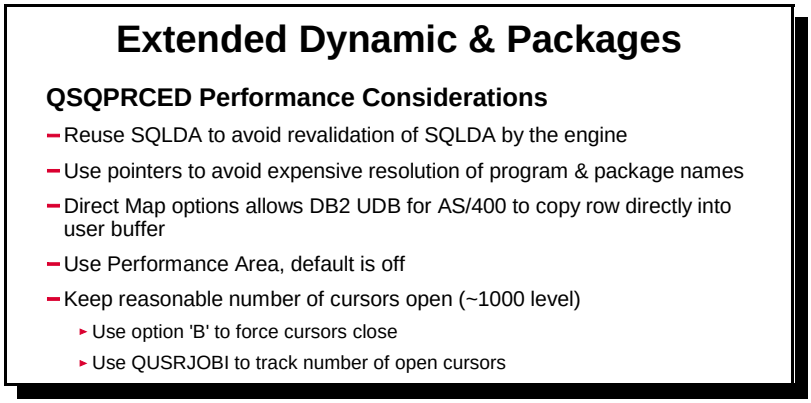
Let’s go one level deeper to assist programmers in better understanding and sizing the effort involved with using the QSQPRCED API. We are not attempting to teach you how to program to this API with these examples; therefore, detailed programming is beyond the scope of this online course. But, additional information on this API, see the Links section at the end of this course.

There is also a Web site that contains FAQs related to improving Performance with SQL Packages listed in the Links section of this course.



Extended Dynamic and Packages — Execute an Insert with QSQPRCED API

Please review this sample code which illustrates the method for executing an insert with the QSQPRCED API. This sample should give you a perspective of the infrastructure required for using this API.



Extended Dynamic and Packages — QSQPRCED Performance Considerations

Some performance considerations and high level suggestions for getting the best productivity when using the QSQPRCED API set include the following:

* Reuse the SQL Descriptor Area (SQLDA) to avoid revalidation of SQLDA by the engine.

* Use pointers to avoid expensive resolution of program and package names.

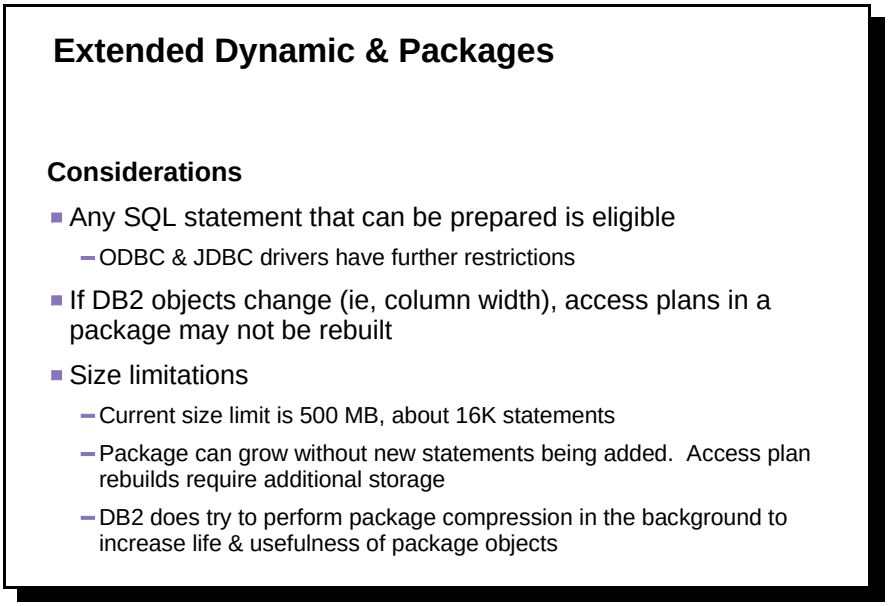
* Use Direct Map options to allow DB2 UDB for iSeries to copy a row directly into the user buffer.

* Use the Performance Area. Remember that the default is set to “off.”

* Maintain a reasonable number of cursors open, somewhere around 1000 is about right.

* Use option 'B' to force cursors to close.

* Use User Job Information (QUSRJOBI) to track the number of open cursors.



Extended Dynamic and Packages — Considerations

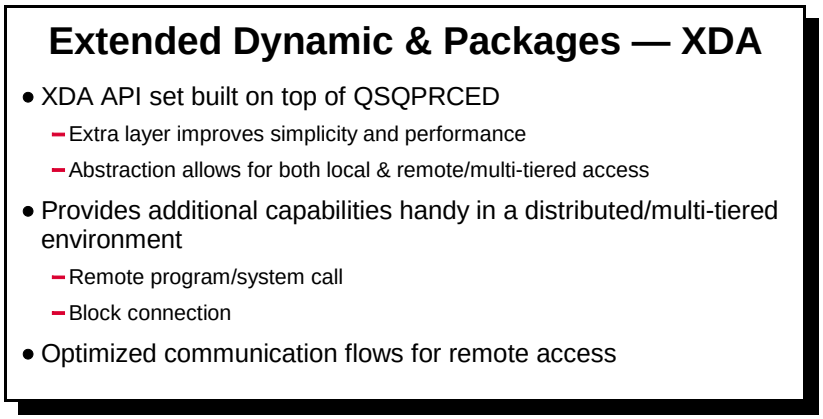
Here are some considerations to keep in mind when using the extended dynamic interface and the associated package.

Any statement that can be prepared (i.e., anything supported by the SQL PREPARE statement) is eligible. But, remember that ODBC and JDBC drivers do has some further restrictions that are discussed in the Coding for SQL Performance online course.

Packages have a maximum size of 500 MB which should accommodate around 16,000 statements. Also, the package size can grow even when new statements are NOT being added. Access plan rebuilds use new storage from the end of the package. The rebuilds are not done in place. DB2 will perform some background processing to reclaim the space for old access plans that were not updated in place. This reclaim processing delays how long it takes to hit the package size limit. Packages become unusable when size limits are encountered, so that's an issue that needs to be considered by the programmer. The iSeries ODBC and JDBC drivers provide an interface for specifying what actions you would like the driver to take when the size limit is reached. Read the documentation for these drivers for more details.

Unlike SQL programs where all access plans are rebuilt when the associated DB2 table definitions are changed, there are times when the plans stored in an SQL Package are not automatically rebuilt by DB2 UDB for iSeries. So, it may be a good idea to recreate your SQL packages any time the definitions for your database objects have changed.

To view some Frequently Asked Questions and Answers regarding SQL Packages, see the Links section of this course.



Extended Dynamic and Packages — XDA

XDA is a set of OS/400 APIs that are built on top of the QSQPRCED API. This API set is primarily used by iSeries solution providers.

This abstraction layer is needed and built for several reasons:

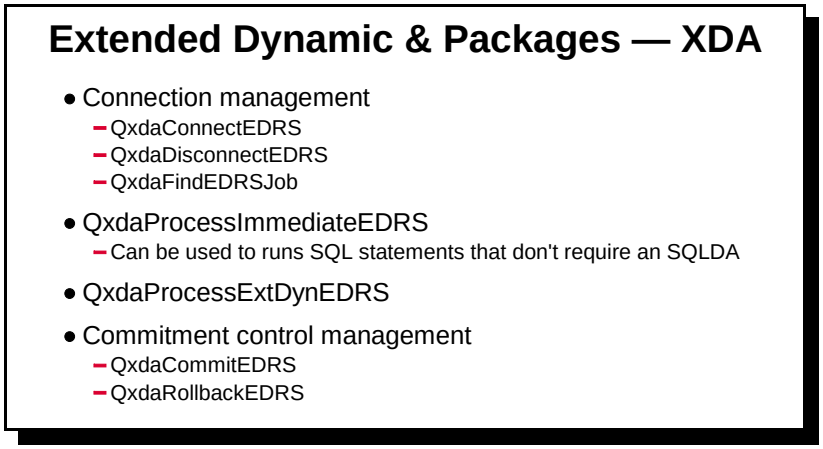
* To provide a generic way of accessing extended dynamic support from a remote client

* To simplify the complexity and management (creating packages) issues associated with QSQPRCED API

* To improve its performance

* To allow for both local and remote multi-tiered access

The XDA API also provides additional capabilities that are handy in a distributed or multi-tiered environment. Remote program and system call is enabled. Additionally, blocked connections allow the connection to be switched from one database server to another. That's very useful in multi-tiered environments.



Extended Dynamic and Packages — XDA (continued)

Here's a list of the core set of XDA APIs. All of the XDA APIs are documented in the OS/400 API manual.

QxdaProcessImmediateEDRS is a good example of the simplification of QSQPRCED. It provides a fast path through the QSQPRCED API for those statements that don't require an SQLDA.

QxdaProcessExtDynEDRS is similar to QSQPRCED. However, it has additional ease of use advantages such as providing the ability to tell QSQPRCED to create the package, caching of SQLDAs to increase SQLDA reuse, etc.

Now you can implement SQL Performance Basics in your sleep.

Conclusion

Congratulations on completing this updated online course on SQL Performance Basics. You should now be familiar with static and dynamic SQL, SQL access plans and ODPs, and dynamic and extended dynamic SQL for DB2 UDB for iSeries. You should also have a good understanding of DB2 UDB for iSeries interfaces that utilize SQL.

You can review any section of the online course at any time to reexamine material with which you want to become better acquainted. You also can download a pdf of this course to use as a reference, along with the many Internet sites that are listed for your perusal.

Thank you for participating in this online course. Good luck as you begin implementing SQL Performance Basics for your organization.

Hotlinks related to this online training course

These Hotlinks may provide additional useful information for you to supplement this course. They provide useful Web sites and reference materials.

- To view additional QSQPRCED coding samples :
  ibm.com/eserver/iseries/db2/db2code.html
- To view **FAQs on Improving Performance with SQL Packages**
  ibm.com/eserver/iseries/db2/sqlperffaq.htm
- iSeries Information Center:
  ibm.com/eserver/iseries/infocenter
- For DB2 UDB SQL Query Engine (SQE) Information
  ibm.com/eserver/iseries/db2/sqe.html
- Additional information on the QSQPRCED API including additional QSQPRCED coding samples
  ibm.com/eserver/iseries/db2/db2code.htm

References and Additional Information

Useful Web sites:

- DB2 UDB for iSeries Web site
  ibm.com/eserver/iseries/db2
- DB2 UDB for iSeries Porting Web site
  ibm.com/servers/enable/site/db2/porting.html
- DB2 UDB for iSeries manuals and documentation
  ibm.com/eserver/iseries/db2/books.htm

Additional Education:

- DB2 UDB for iSeries SQL and Query Performance Tuning & Monitoring Workshop
  ibm.com/eserver/iseries/service/igs/db2performance.html

Online courses, Downloadable Labs, & White Papers:

- **DB2 UDB for iSeries SQL Performance & Query Optimization**
  ibm.com/servers/enable/site/education/ibo/view.html?oc#db2
  ibm.com/servers/enable/site/education/ibo/view.html?wp#db2

Books:

- SQL/400 Developer's Guide by Paul Conte & Mike Cravitz
  http://as400network.com/str/books/Uniquebook2.cfm?NextBook=183
- SQL for eServer i5 and iSeries by Kevin Forsythe
  www.mc-store.com/5063.html

Trademarks

This publication may have referred to products that are not available in your country. IBM, eServer, iSeries, OS/400, DB2, and DB2 Universal Database are registered trademarks of the IBM Corporation in the United States or other countries or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States and other countries.

Microsoft, Windows and the Windows logo are trademarks or registered trademarks of Microsoft Corporation.

IBM makes no commitment to make available any products referred to herein.

All other trademarks and registered trademarks are the properties of their

Test Questions

1. SQL Precompilers allow you to embed SQL in most of the popular iSeries programming languages, including:

   ❑ RPG and C/C++
   ❑ COBOL
   ❑ SQL Stored Procedure Language
   ☑ All of the above

2. Dynamic SQL interface constructs SQL statements:

   ❑ Within the optimizer
   ☑ On the fly
   ❑ As required for system execution
   ❑ Concatenated inside the embedded code

3. DB2 UDB for iSeries interfaces that utilize Dynamic SQL include:

   ❑ JDBC and ODBC
   ❑ CLI and Net.Data
   ❑ Interactive SQL
   ☑ All of the above

4. For Static SQL, an Access Plan is created for each SQL statement and:

   ☑ Stored into the program object
   ❑ Validated by the precompiler
   ❑ Eventually slows the system if unchecked
   ❑ Stored in working memory

5. Access Plan contents include:

   ❑ Access methods
   ❑ Information on associated tables and indexes
   ❑ Applicable program and/or environment information
   ☑ All of the above

6. Access Plan contents vary between the static and dynamic SQL requests.

   ❑ True
   ☑ False

7. Access plans for Dynamic SQL requests are stored in permanent objects.

- ❑ True
- ☑ False

8. Once the access plan is built, which of the following methods can be used to rebuild the access plan?

   - ❑ End user submits a change request form
   - ❑ Programmer or administrators execute a BIND request
   - ❑ Reinstallation of DB2 UDB for iSeries
   - ☑ DB2 UDB for iSeries automatically rebuilds the access plan when the environment or database changes

9. You cannot pre-open ODPs with SQL.

   - ☑ True
   - ❑ False

10. Which of the following is an iSeries technique that can help minimize negative performance impacts?

    - ❑ Shared access plans
    - ❑ Reusable ODPs
    - ❑ Extended dynamic SQL
    - ☑ All of the above

11. Creating an ODP requires _____ more system resource than reusing an existing ODP.

    - ❑ 50%
    - ☑ 10 to 20 times
    - ❑ 25%
    - ❑ 50 times

12. DB2 UDB for iSeries attempts ODP reuse when:

    - ❑ Encountering an SQL REUSE statement in the application
    - ❑ Specifying the precompiler reuse option
    - ❑ The same SQL statement has been executed more than twice on the system
    - ☑ The same SQL statement has been executed more than twice within a job

13. Pseudo-close refers to:

- ❑ Performing a sales transaction without totaling the account's bill
- ❑ Moving ODPs into the recycle bin, but not emptying the bin
- ❑ Maintaining ODPs in an "inactive" status
- ☑ The ODP and cursor being closed, but remaining active

14. ODPs can be shared and reused by different jobs.

- ❑ True
- ☑ False

15. Roadblocks to reusing ODPs can include:
- ❑ View Materialization
- ❑ Creation of temporary tables
- ☑ Permitting unqualified SQL tables in your SQL requests to run with system naming mode (*SYS)
- ❑ None of the above

16. Temporary indexes always cause the associated ODP to be non-reusable.
- ❑ True
- ☑ False

17. Temporary indexes can be shared by other ODPs.
- ❑ True
- ☑ False

18. Reusable ODPs have one shortcoming:
- ❑ They may wear out after multiple IPLs
- ❑ They may not maintain benchmark response times
- ☑ In reusable mode, DB2 UDB for iSeries cannot react to environment changes and rebuild the access plan
- ❑ Access Plans are constantly rebuilt

19. With Dynamic interfaces, a good rule to remember in avoiding FULL OPENS is:Always keep background caching out in the open
- ❑ To use automatic PREPAREs for DB2 UDB for iSeries
- ☑ "Prepare once, execute many"

20. A PREPARE automatically creates a new statement and full open on each execution.
- ❑ True
- ☑ False

21. _____ are an effective way of making dynamic SQL requests, and the associated ODPs, easier to reuse.
    - ❑ SQL CONNECTs
    - ❑ Automatic PREPAREs
    - ☑ Parameter markers
    - ❑ SQL Optimizers

22. The CAST scalar function allows parameter markers to be used in more places:
    - ❑ Because the optimizer makes all choices for implementation
    - ☑ Since CAST can be used to define the parameter marker attributes for DB2 Since without CAST the parameter markers become transparent
    - ❑ However, a CAST may impede getting outside the parameter
    - ❑ None of the above

23. Dynamic SQL requests can be run in extended dynamic mode by just changing the appropriate iSeries job attribute.
    - ❑ True
    - ☑ False

24. In Extended Dynamic SQL, search of the package is accomplished by:
    - ❑ Asking known users to voluntarily submit to searches
    - ❑ Requesting to review a user's ODPs
    - ❑ Mirroring how the Acess Plan for embedded static SQL is deleted
    - ☑ Statement text to determine if the package contains an access plan that can be reused for the SQL request

25. What is an advantage of using Extended Dynamic SQL Packages:
    - ❑ The package can share statement access plan information to all systems users.
    - ❑ The package is permanent across job and system termination.
    - ❑ Improved performance decisions since statistical information is accumulated for each SQL statement.
    - ☑ All of the above

26. What are the interface(s) into iSeries extended dynamic support?
    - ❑ A system API
    - ❑ The XDA API set
    - ❑ An option on the IBM provided ODBC and JDBC drivers
    - ☑ All of the above

27. _____ tries to reduce the complexity and burden from the QSQPRCED API.
    - ❑ A system API
    - ❑ The XDA API set
    - ❑ An option on the IBM provided ODBC and JDBC drivers
    - ☑ All of the Above