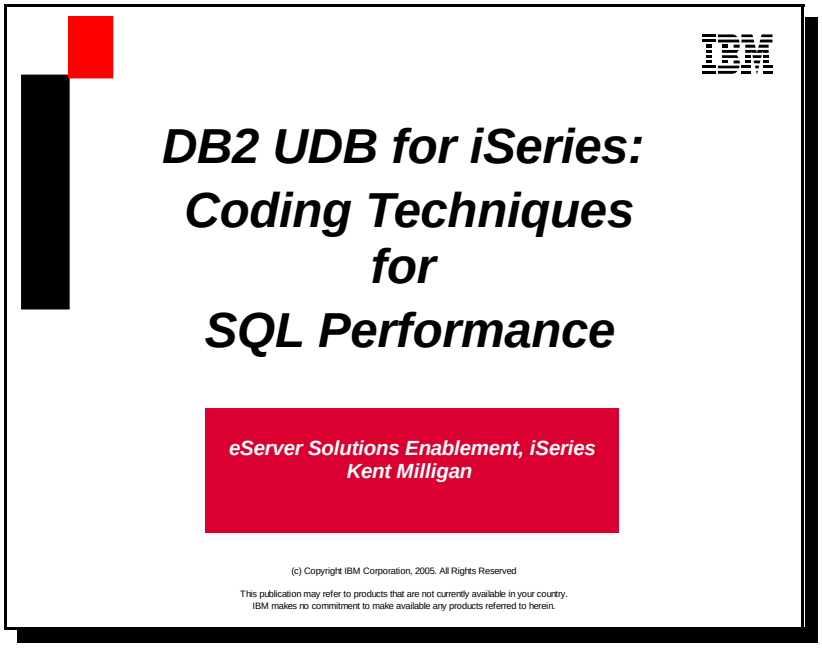
## Contents




# DB2 UDB for iSeries: Coding Techniques for SQL Performance

## Instructions

### Requirements

- Java is required for the audio
- JavaScript is required to navigate

Click here to test your browser for compatibility.

**Browsers**
If you have one of these browsers you should be set.

- Netscape Communicator/Navigator 4.0.5 or better
- Microsoft Internet Explorer 4.0 or better

**You can download this course and take it off-line**
This course can be downloaded so you can conveniently take it off-line. You will need to have installed a copy of Adobe Acrobat Reader (you can download this reader by clicking here). If you want to download the Acrobat files related to this course click here. There are two separate files to download. One contains the foil screens (graphics), the other contains the speaker notes (text). The foil and speaker notes pages have been numbered to help you in putting them together.

**Hotlinks**
As you go through this course, you will observe, and probably take advantage of, frequent hotlinks that send you to other Web sites for more details about a particular topic. These hotlinks have been compiled in one long list at the end of the course, along with enough of a descriptor to remind you what they are about, and can be accessed by clicking on the LINKS option in the navbar.

We think these links are important, but you may prefer to visit these sites at some later point so that, for right now, you can stay focused on the “meat” within this course. You can clip the contents of this “hotlinks” page into a document file on your local PC drive. Then, at your convenience, you can use these saved hotlinks to explore these topics in greater detail.

The Hotlinks page also has other Web site and reference information that pertains to this course.

**Audio**
All course pages in this IBE include streaming audio. You will see a button that looks like this in the lower left corner of the course window.

The audio clips will automatically start playing, so you only need to press this button if you want to pause or replay the clip.

**Speaker Notes**
Open or close the Speaker Note page with these buttons.

**Navigation:**
These are the general navigation buttons:

- Click NEXT to go forward one page.
- Click PREV to go back one page.
- The TOPICS page lists all the pages in this IBE for quick navigation.
- The LINKS page contains the hotlinks to related information.
- The FAQ page lists frequently asked questions about the subject.
- The QUIZ page asks you questions about what you've learned.
- Click EXIT to leave the course.

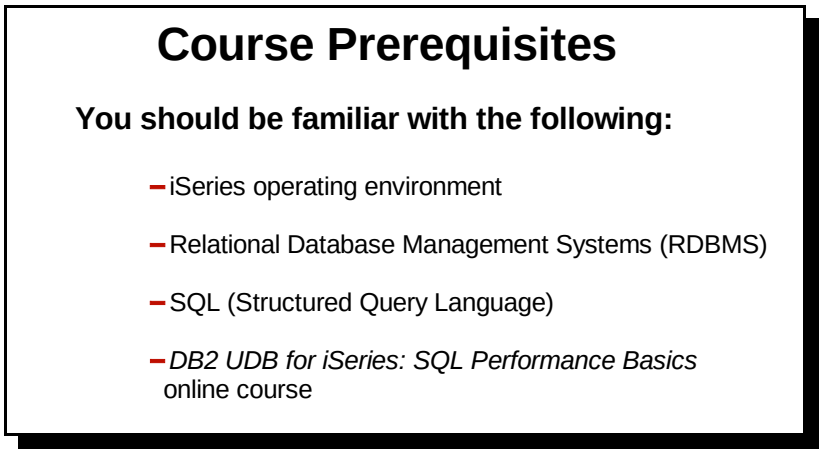
## Author Biographical Information

**Kent Milligan, DB2 UDB Technology Specialist**
**eServer Solutions Enablement, iSeries**

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



.

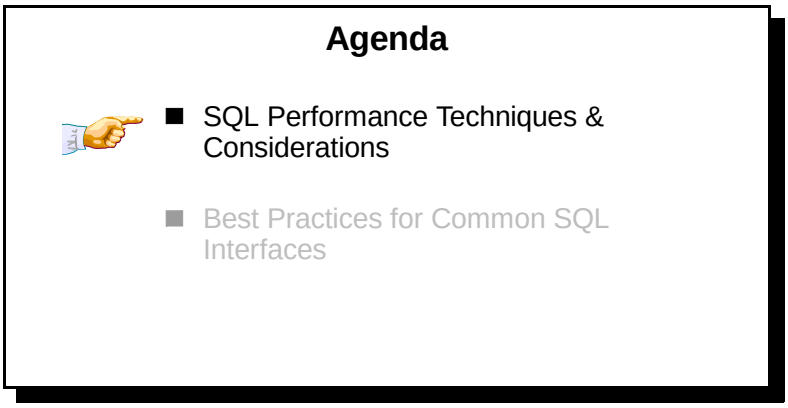



## Course Prerequisites

Hello, and welcome to this online course, *DB2 UDB for iSeries: Coding Techniques for SQL Performance*! We'll make a few assumptions as you begin. You should have a basic understanding of the IBM® eServer™ i5 and IBM eServer iSeries™ operating environment. We also assume you are already familiar with basic IBM terminology, and therefore, these terms will not be further explained in this learning course (except where noted). You should also be familiar with Relational Database Management Systems (RDBMS) and SQL, and have taken the *DB2® UDB for iSeries: SQL Performance Basics* Internet-based education course, which can be found at the Web site provided in the Links section of this course.

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 that segment, you will be given the opportunity to return to the latest segment of the course you were viewing.

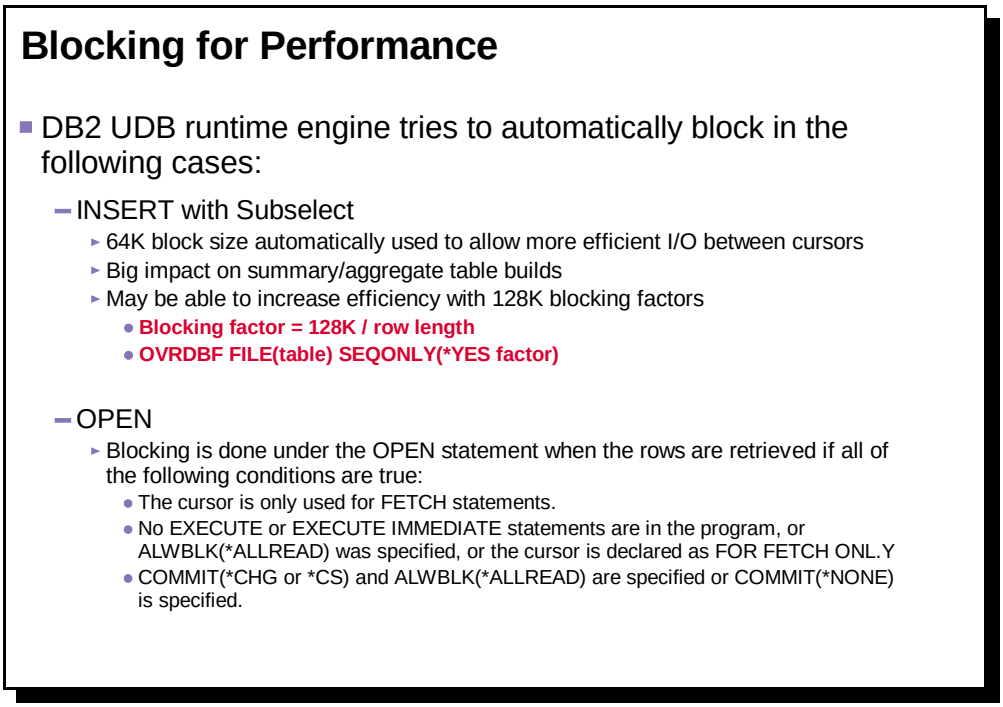
You will also be occasionally quizzed on the material you have learned, so that you can get a sense for your level of understanding before moving on to the next segment. Then, at the end of the course, you will be asked to take a final test regarding this material.




## Agenda: SQL Performance Techniques & Considerations

This course will cover SQL performance techniques and considerations and the best practices for common SQL interfaces as they relate to DB2 UDB for iSeries.

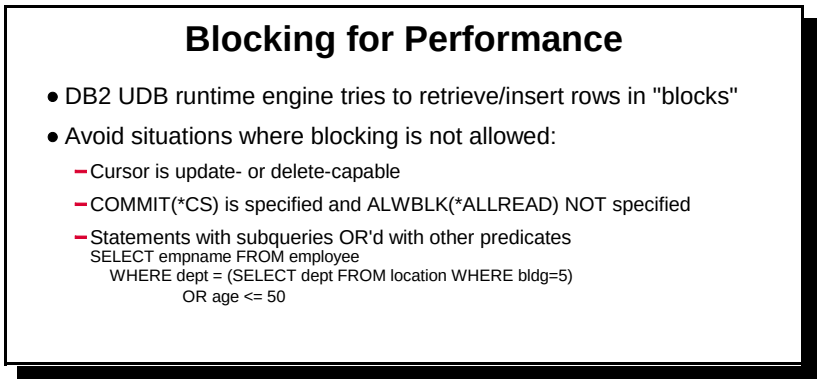
First, let's talk about good SQL programming techniques and other factors that impact the performance of SQL within the DB2 UDB for iSeries environment.




## Blocking for Performance

DB2 UDB for iSeries will automatically attempt to increase its blocking factor for the ‘Open’ and ‘Insert with Subselect’ statements. This effort is based on the conditions listed on this chart.

If the ‘Insert with Subselect’ statement is putting millions of rows into the target table, then it may be worth the programming effort of manually setting the blocking factor to 128 kilobytes through the use of the Override with DataBase File (OVRDBF) command. The reason for this is that DB2 UDB for iSeries will only adjust the blocking factor up to 64 kilobytes  for the ‘Open’ and ‘Insert with Subselect’ statements.



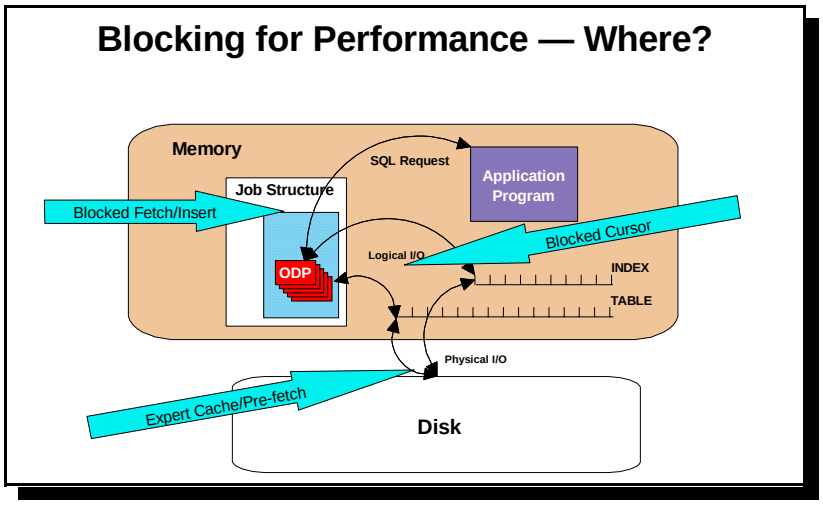

## Blocking for Performance

The DB2 UDB for iSeries runtime engine attempts to retrieve and insert rows by managing the data as “blocks” whenever possible to reduce the number of read/write efforts. However, there are a few situations when blocking is not permitted.

Because blocking offers so much efficiency, you should understand, and avoid as much as possible, those situations where blocking is not permitted. Let’s take a look at some examples of when blocking is not allowed:

- Blocking is not allowed when the cursor is update- or delete-capable.
- Blocking is not done when COMMIT (*CS) is specified and ALWBLK (*ALLREAD) is NOT specified.
- Blocking is not performed when the length of the result set size is greater than 32 kilobytes.

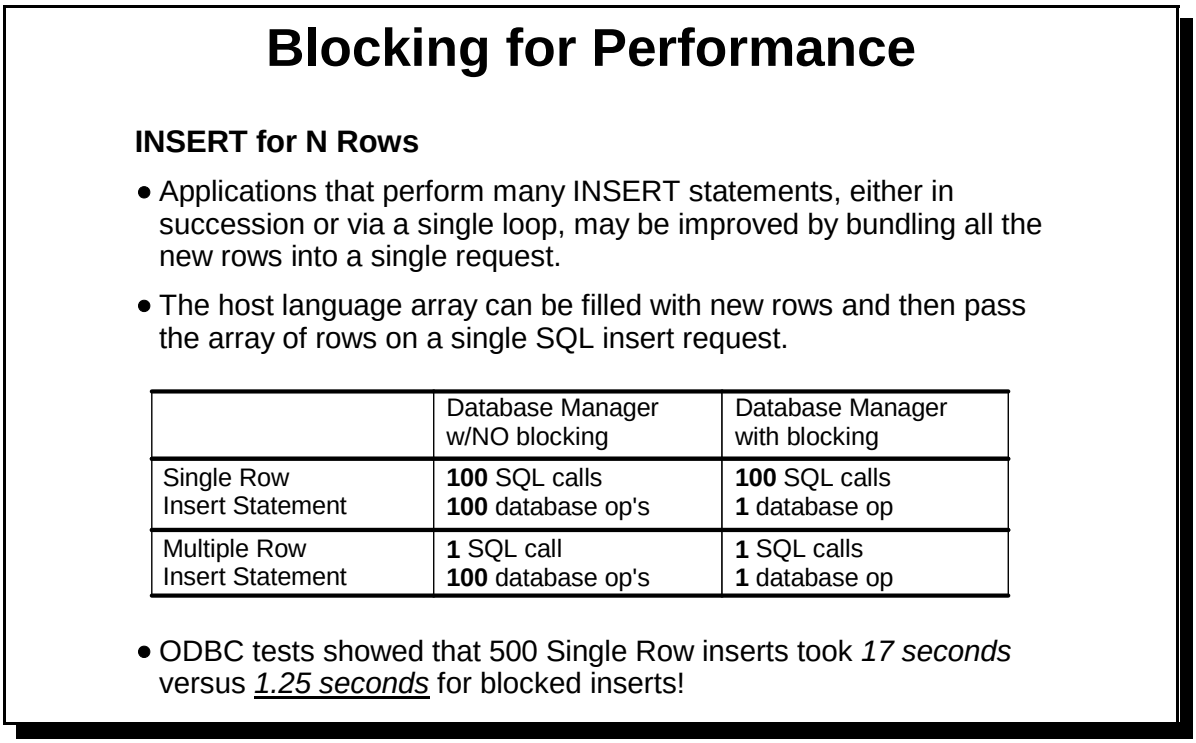
.




## Blocking for Performance — Where?

You can see that this diagram illustrates the places where blocking can occur. The system automatically blocks in the background through the utilization of expert cache and the pre-fetching of rows. (Note that the expert cache option is activated by setting the memory pool option to *CALC.)

The SQL application/requester can also force the use of blocking by utilizing blocked insert and fetch constructs in the application.



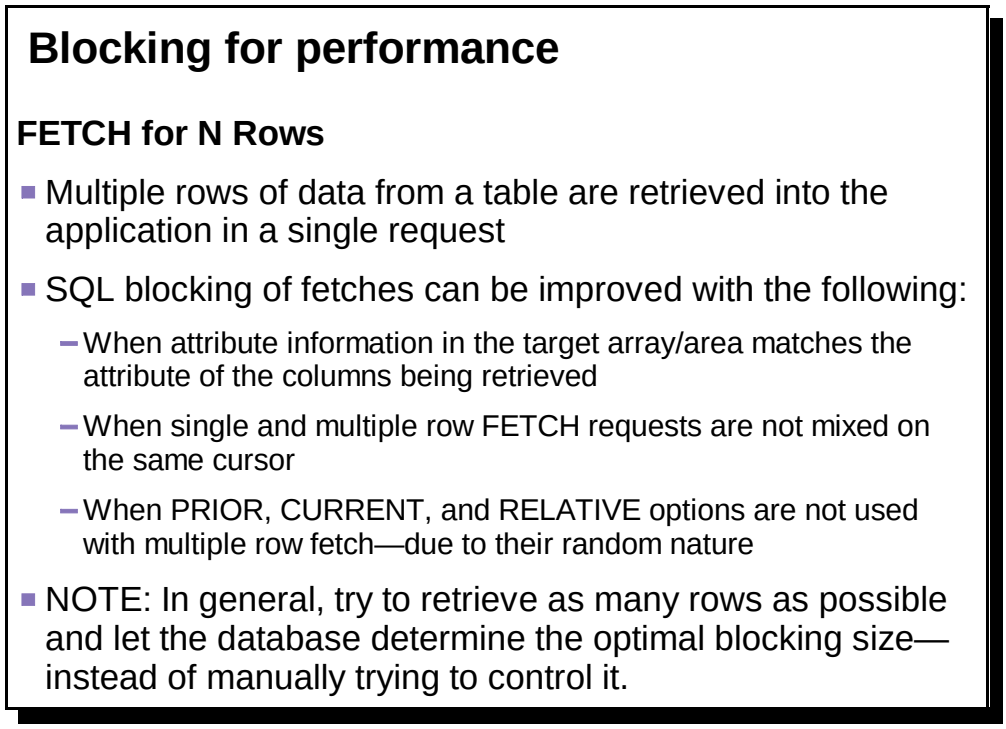

## Blocking for Performance — INSERT for n Rows

The table on this chart shows that much data access effort can be eliminated by blocking at the application and database engine levels. Here's how:

In a single row insert statement case that involves 100 SQL calls, DB2 UDB for iSeries, using no blocking, will require 100 database operations. With blocking, the same 100 SQL calls will require only one database operation — thereby using much less system resource.

In the case of multiple row insert statements that involve 100 SQL calls, DB2 UDB for iSeries requires 100 database operations for one SQL call if no blocking is used. However, with blocking, the multiple row insert statements require only one database operation for one SQL call — yet 100 new rows will have been inserted.

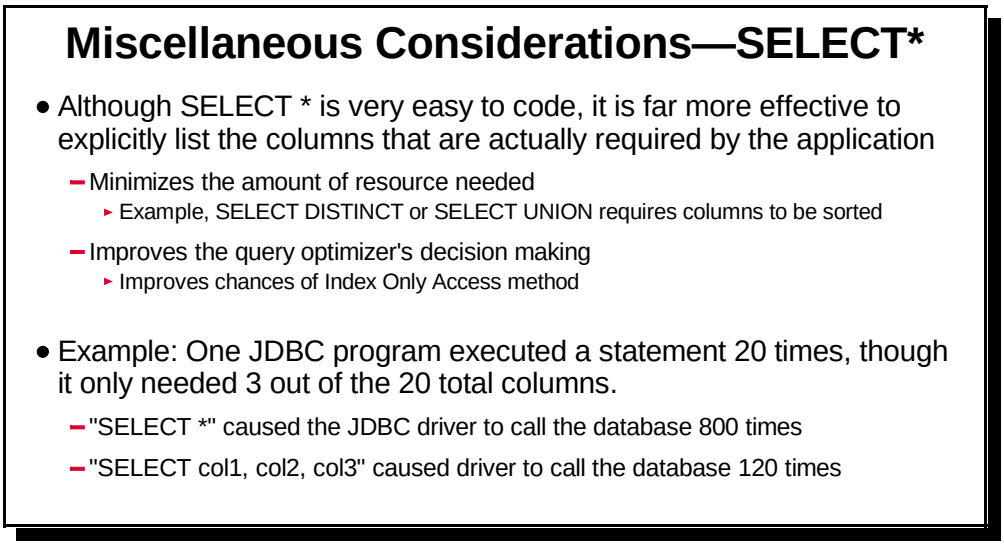
ODBC tests have clearly revealed the benefits of bundling database requests. For example, in one test, ODBC performance went from 17 seconds down to 1.25 seconds when database requests were bundled. That's a 93% improvement in performance that results from bundling requests for DB2 UDB for iSeries.




## Blocking for Performance — FETCH for n Rows

By using “FETCH for n rows”, multiple rows of data can be retrieved into an application by a single request. This chart shows that SQL blocking of fetches can be improved most efficiently when:

- The attribute information in the target array/area matches the attribute information of the columns being retrieved.
- You do not mix single and multiple row FETCH requests on the same cursor.
- And, you do not use PRIOR, CURRENT and RELATIVE options with multiple row fetches, due to the random nature of accessing rows that these options invoke.




## Miscellaneous Considerations — SELECT*

We'll cover several suggested coding practices for DB2 UDB for iSeries under "miscellaneous considerations" as you study this and the next six charts.

First, consider avoiding SELECT * when blocking for performance. Although SELECT* is very easy to code, the optimizer will have a better chance of selecting "index only access" if you specifically list the columns required by the application. Thus, all the columns can participate in a sort operation.

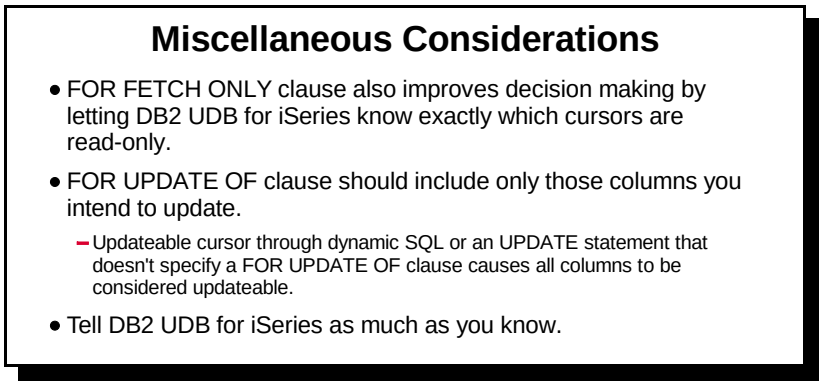
Listing the columns that are required by the application:

- Minimizes the amount of resource needed and the amount of data moved. For example, SELECT DISTINCT or SELECT UNION requires that the columns be sorted — if all of the columns are selected with *, then all the columns will be sorted.
- Improves the query optimizer's decision making, thus improving the chances of the "index only access" method, by providing the optimizer with as much information as possible about your request and data.

Let's consider the example of a JDBC program that executes a statement 20 times, even though it actually only requires data from three of the 20 total available columns.

- SELECT * causes the JDBC driver to call into the database engine 800 times.
- In contrast, SELECT col1, col2, col3 will cause the driver to call into the database only 120 times.

This reduces demand on the database by 85%, so you can easily see that the DB2 UDB for iSeries resource is greatly reduced when you specifically list only those columns that are required by the application, rather than simply coding SELECT *.



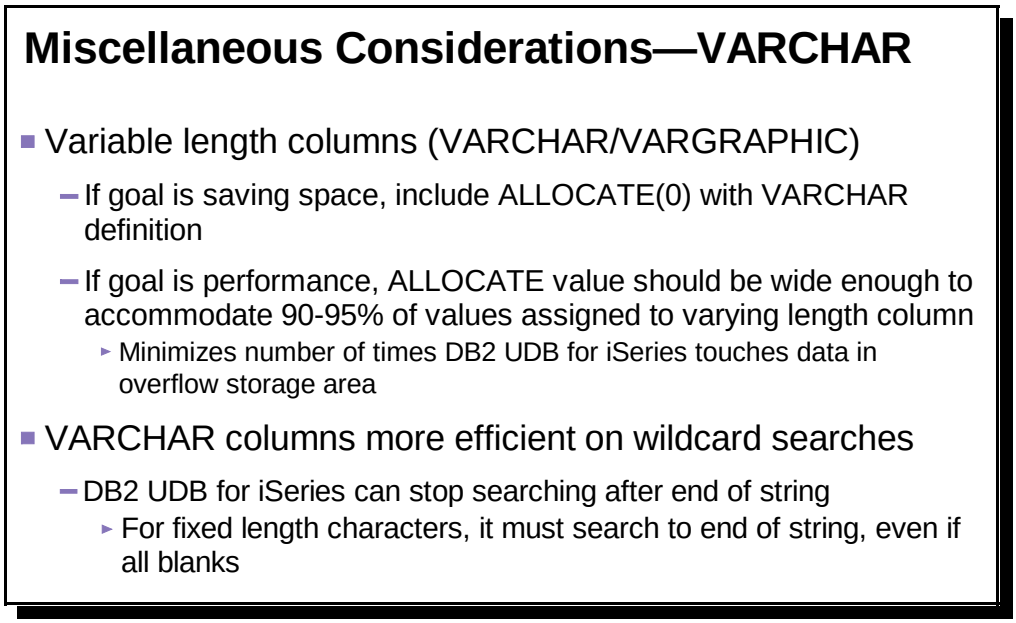

## Miscellaneous Considerations

Using the FOR FETCH ONLY clause and the FOR UPDATE OF clause helps the optimizer better understand your SQL request. This is because these two clauses let DB2 UDB for iSeries accurately know which cursors are read-only.

Please remember, when you use the FOR UPDATE OF clause, only include the columns you actually intend to update. Why? Because an updateable cursor through dynamic SQL, or an UPDATE statement that doesn't specify a FOR UPDATE OF clause, causes all columns to be considered updateable.

A good rule to follow is to "code as much as you know" so that DB2 UDB for iSeries will have all the information it needs to choose the best performing method.

OPTIMIZE FOR n ROW clauses and WITH DISTINCT VALUES clauses are other examples of providing additional, useful information for DB2 UDB for iSeries to run your SQL more efficiently.



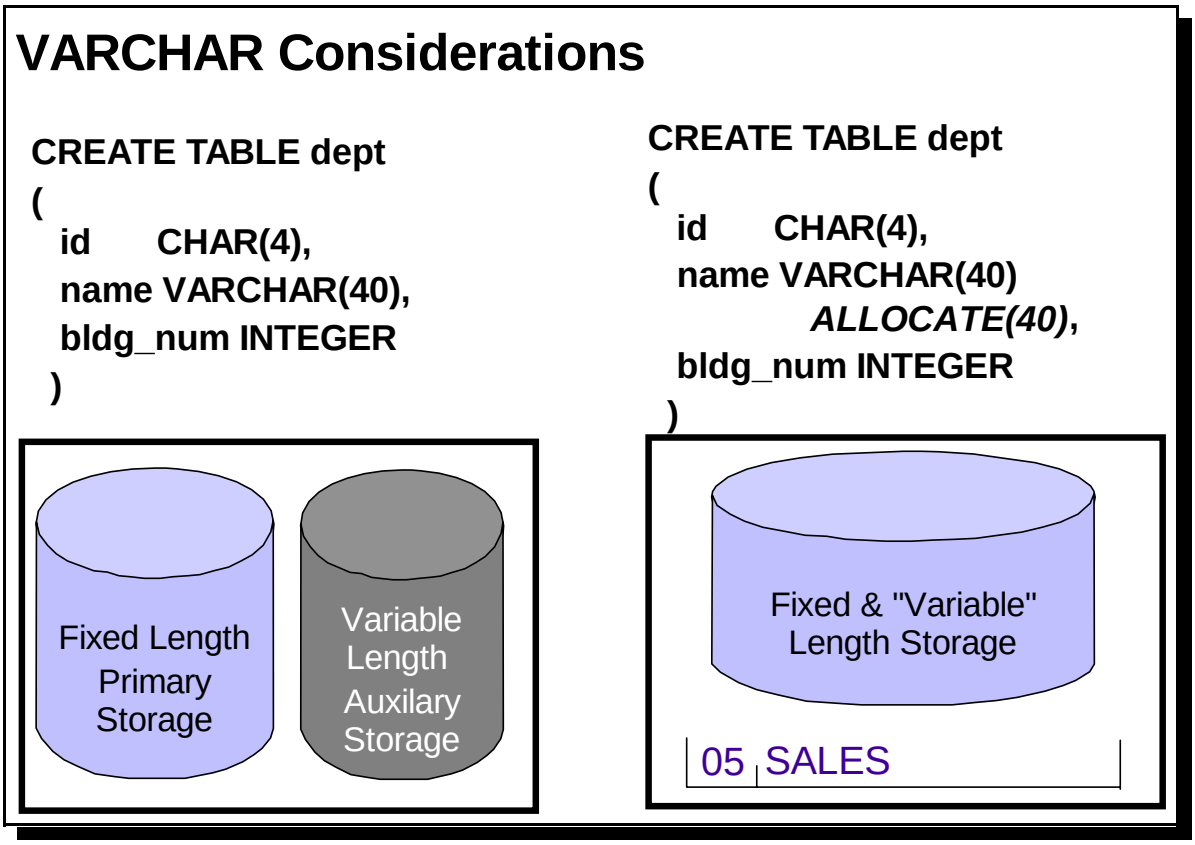

## Miscellaneous Considerations — VARCHAR

Even though the use of variable length columns (VARCHAR and VARGRAPHIC) is popular on other databases, this technique may not be the best coding method for DB2 UDB for iSeries. Let's carefully consider the use of VARCHAR columns in DB2 UDB for iSeries.

If your primary goal is saving disk space, then set the ALLOCATE setting for a VARCHAR column to "0" when defining VARCHAR. In fact, if you don't specify the ALLOCATE setting, it will automatically default to "0".

However, if you are more concerned about improving performance than saving space, you will instead want to set the ALLOCATE value wide enough so that it can accommodate 90 to 95 percent of the variable length values found within that column. This will minimize the number of times that DB2 UDB for iSeries must examine the data stored in the overflow area of iSeries. On the next chart you'll see a picture of how the storage of VARCHAR column changes with the ALLOCATE keyword.

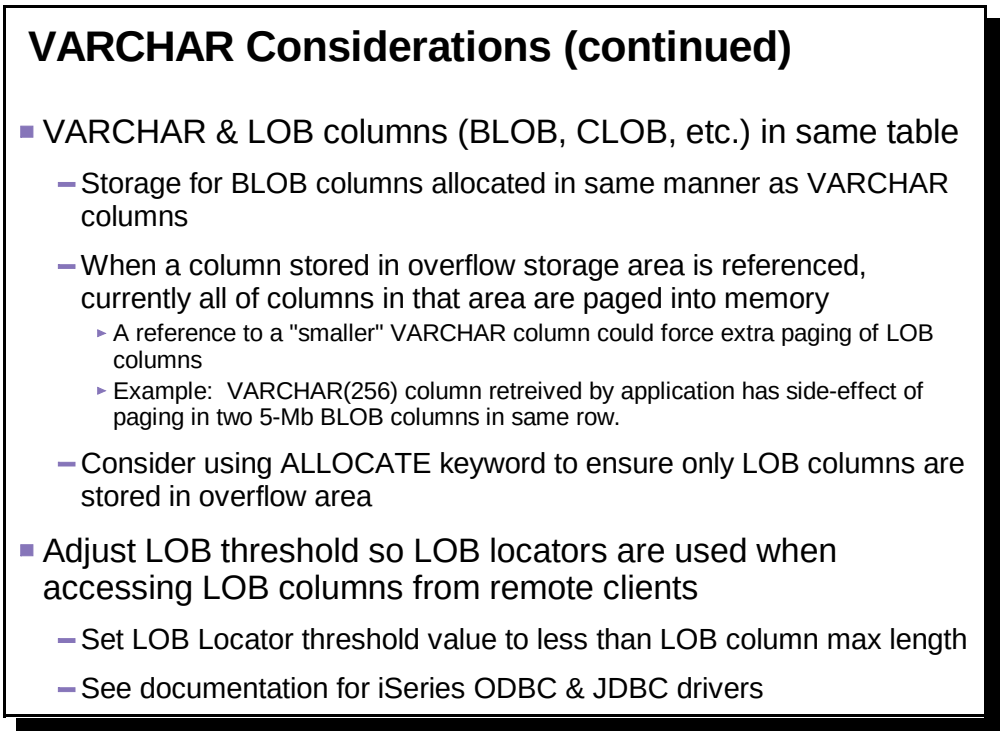
VARCHAR columns are, however, more efficient on iSeries when there is a need to provide for wildcard searching. That's because DB2 UDB for iSeries will stop searching VARCHAR columns after a wildcard string search encounters blanks that indicate the end of available characters. With fixed-length characters, DB2 UDB for iSeries must search to the very end of the string, even if the string contains blanks.




## VARCHAR Considerations (continued)

The example on the left shows the default storage behavior of VARCHAR columns where the name column is stored in the auxiliary storage container. On the right side, you can see how the ALLOCATE keyword eliminates the need for the extra storage container — which means fewer I/O requests at runtime since DB2 UDB for iSeries only has to worry about paging in a single storage container.

Internally, DB2 UDB for iSeries will automatically set the Allocate value to the maximum length of the variable length column whenever the column length is 30 bytes or less.

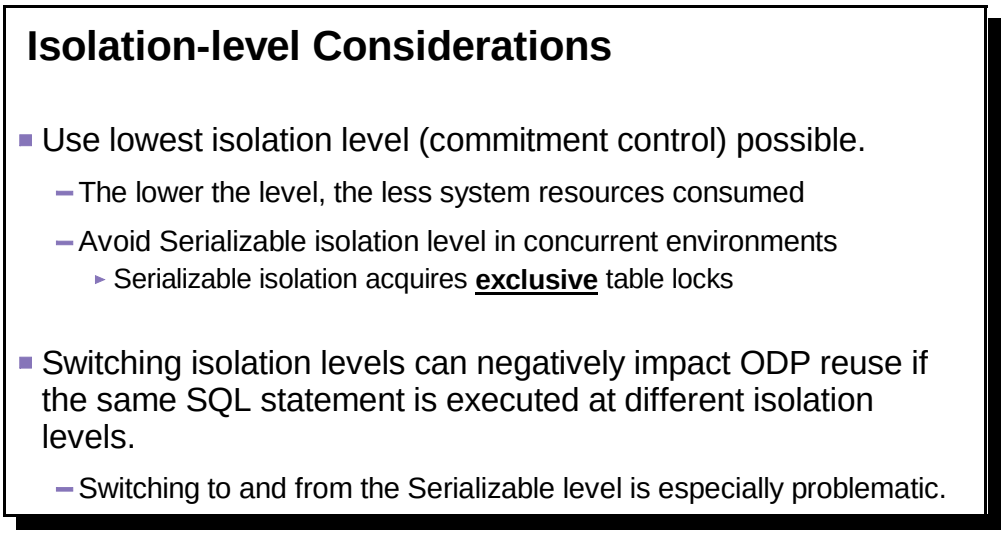



## VARCHAR Considerations (continued)

You should also be aware that VARCHAR columns with a small allocate value can cause performance problems if the table also contains LOB (large object) columns and the VARCHAR column is referenced frequently. The VARCHAR and LOB columns are all stored in the same auxiliary overflow storage area. This means that if you reference one of the columns in that area, the database has to page in all the overflow storage columns. In general, DB2 UDB for iSeries tries to avoid paging in large LOB columns, but it can't be avoided when the row contains VARCHAR columns that are stored in the same auxiliary storage area.

If LOB columns are accessed by remote clients via middleware, such as ODBC or JDBC, then the LOB Locator Threshold setting can impact performance. The use of locators minimizes the amount of LOB data that DB2 UDB for iSeries will copy to the remote client. When locators are not used (Threshold greater than column max length), than DB2 UDB for iSeries currently always copies the maximum length of the LOB column instead of just the actual data length. For instance, if a 2-megabyte LOB column contains a LOB value that's 200 kilobytes in actual length, than without locators, DB2 UDB for iSeries will copy two megabytes of data to the requester — instead of just the 200 kilobytes containing the actual data.

.




## Isolation-level Considerations

You can conserve DB2 UDB for iSeries resources by using the lowest isolation level (or commitment control level) within your application. That's because fewer system resources are needed to implement the requested lower level. .

Switching isolation levels within a connection sometimes causes Open Data Paths (ODPs) to be deleted unnecessarily, which can impact performance.

## Journal Considerations

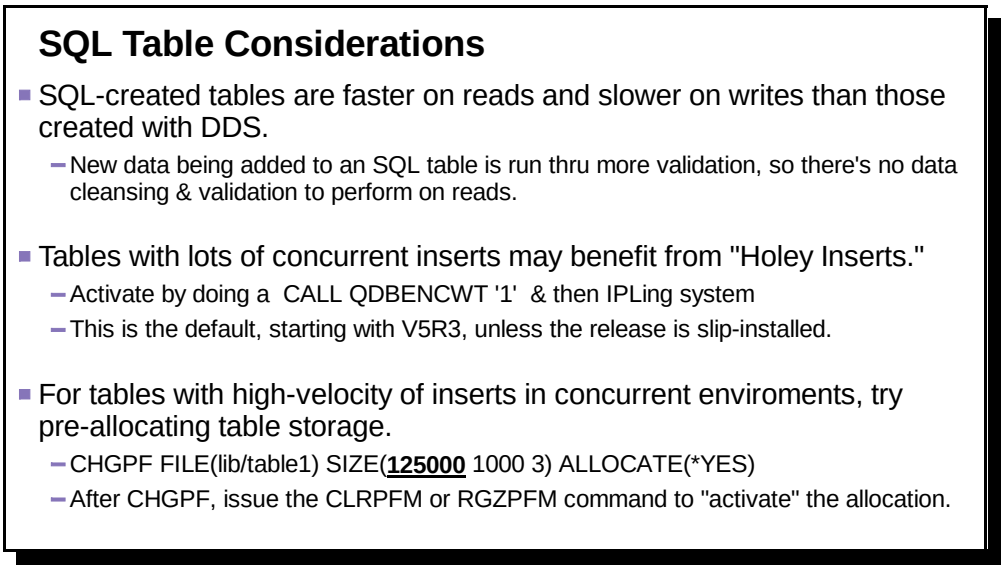
- DB2 UDB for iSeries tries to journal (log) all SQL-created tables automatically when tables are created in an SQL Schema/Collection.
  - Verify that tables are only journaled when required.
- Journals can definitely impact on SQL performance, so that's another area of investigation when doing database performance analysis. Possible places to start:
  - Journal minimal data option to minimize amount of data copied into the journal and size of the journal object
    - MINENTDTA Option on CRTJRN & CHGJRN CL commands
  - Journal Caching PRPQ (5799-BJC) if running batch jobs with isolation level of No Commit/*NONE
  - Hardware configuration: Look for limited Write Cache
  - Redbook: Striving for Optimal Journal Performance (SG24-6286)

## Journal Considerations

DB2 UDB for iSeries users need to be aware that SQL will automatically try to journal a table. For instance, all tables created into an SQL collection are automatically journaled. You should verify that your DB2 UDB for iSeries tables are only journaled when the application requires it, since journaling does put a demand on system resources.

Journaling is used on many iSeries servers, but as is true with any technology, performance tuning is needed.

The Redbook, *Striving for Optimal Journal Performance*, contains lots of information on the best ways to tune an application that is using journaling. A link to the IBM Redbooks Web site is providing in the “Links” section of this course.

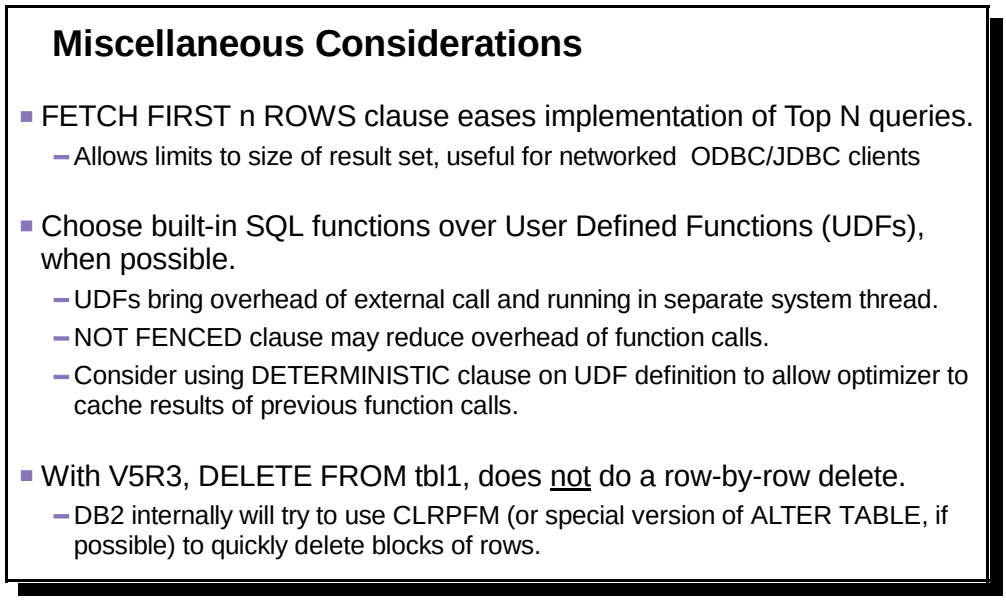



## SQL Table Considerations

SQL-created tables in DB2 UDB for iSeries are faster on reads, and slower on writes, than Data Description Specification-based (DDS) tables. This is because new data being added to SQL tables is run through more data validation initially. Therefore, there's no data cleansing, or validation that is required on reads. (It is true, however, that less data checking is done on inserts into DDS-created tables.)

The capability of utilizing "holey Inserts" is a feature that can improve performance for those DB2 tables that are experiencing a high-degree of concurrent insert (or write) activity. When "holey inserts" is not activated, DB2 UDB for iSeries serializes and only allows one writer at a time. This behavior prevents holes or unused record slots in the table because the next writer is not given a record slot until the current writer has successfully completed its insert operation. With "holey inserts" activated, the database engine allows multiple writers to perform inserts concurrently. This improves performance in concurrent environments at the expense of having some unused record slots (or holes) in the middle of the table. If the ‘Reuse Deleted Records’ feature is being used (this is the default for SQL tables), then DB2 UDB for iSeries will eventually use these holes on future insert operations.

If you have tables where new rows are being inserted at high rates by multiple connections and jobs, especially in batch processing environments, then you may want to consider pre-allocating storage for the rows in those tables when you have a rough idea on the maximum row count for a table. The reason for pre-allocating the table storage is so that DB2 UDB for iSeries can just insert new rows into the table during heavy, concurrent loads instead of periodically having to allocate storage for new rows, interrupting the insertion process. If lots of connections and jobs are allocating new space for a DB2 table at the same time, then the queue for storage allocation can quickly become a bottleneck.

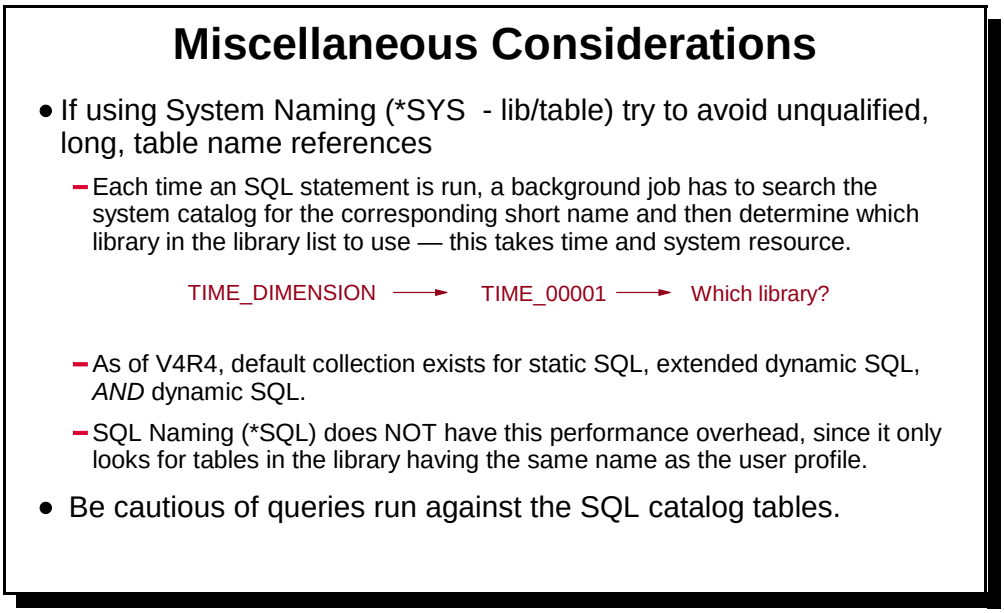



## Miscellaneous Considerations

The FETCH FIRST n ROWS clause can be added to make it easier to implement "Top N" types of reports — such as returning a list of the top ten selling products. In addition, it can be used to limit the size of a result set returned over the network. Less data being sent over the network means improved network performance.

If you have a choice between using the built-in functions of SQL or the User Defined Functions (UDFs), always choose the functions built into SQL. That's because UDFs bring their own resource overhead by requiring external calls, and running in separate system threads. Therefore, if some activity can be implemented with the system's built-in functions, and performance is crucial, UDFs should not be used. If the SQL Query Engine (SQE) is used to process an SQL statement referencing a UDF, then specifying the NOT FENCED clause on the UDF definition can significantly improve the performance of UDF calls. Starting with V5R3, SQE also has the ability to cache the results of UDFs that are declared with the DETERMINISTIC clause. If this type of UDF is called multiple times with the same input parameters, SQE can return the cached results without incurring the overhead of invoking the UDF.

DELETE statements that do not have any selection (i.e., no WHERE clause) will also be faster as the result of another V5R3 enhancement. In V5R3, DB2 UDB for iSeries will attempt to use the Clear Physical File (CLRPFM) command or a special version of ALTER TABLE to delete blocks of rows instead of deleting one row at a time, as it did in previous releases.



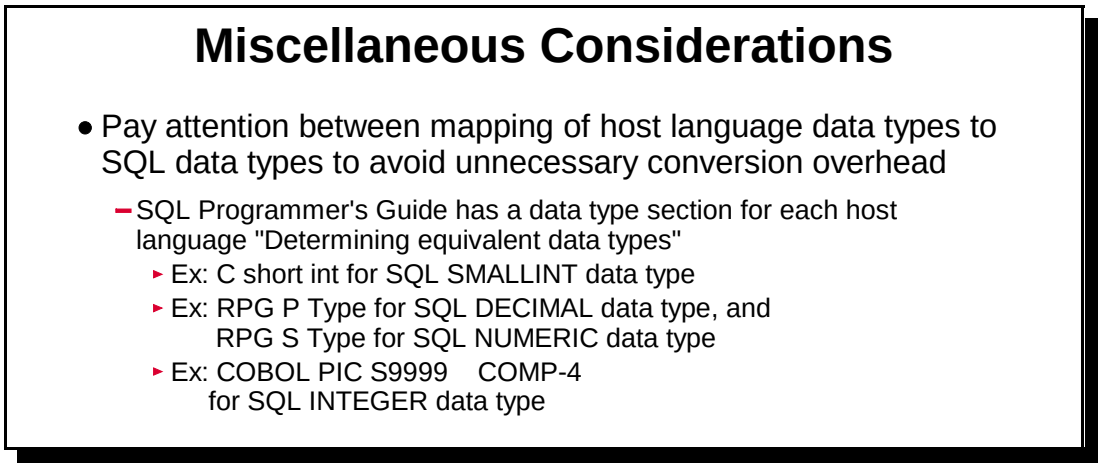

## Miscellaneous Considerations — System Naming Mode (*SYS)

Avoiding unqualified, long table name references when using the System Naming mode (*SYS-lib/table) is a good coding practice in DB2 UDB for iSeries. If you do not avoid these types of references, then DB2 UDB for iSeries will have to execute a background search of the system catalog to find the table's short name. The short name is needed to determine which library within the library list to use — OS/400 does not allow the library list to be searched with the long name. In turn, queries run against system catalogs can be very inefficient. This is because a large percentage of the objects on the system are actually database objects, so the catalogs tend to get large in a hurry.

You can use the default collection precompiler option to eliminate this performance overhead on DB2 UDB for iSeries. Please note that, as of V4R4, default collection is available for all flavors of SQL. Prior to V4R4, default collection was only available for static and extended SQL, but not for dynamic SQL.

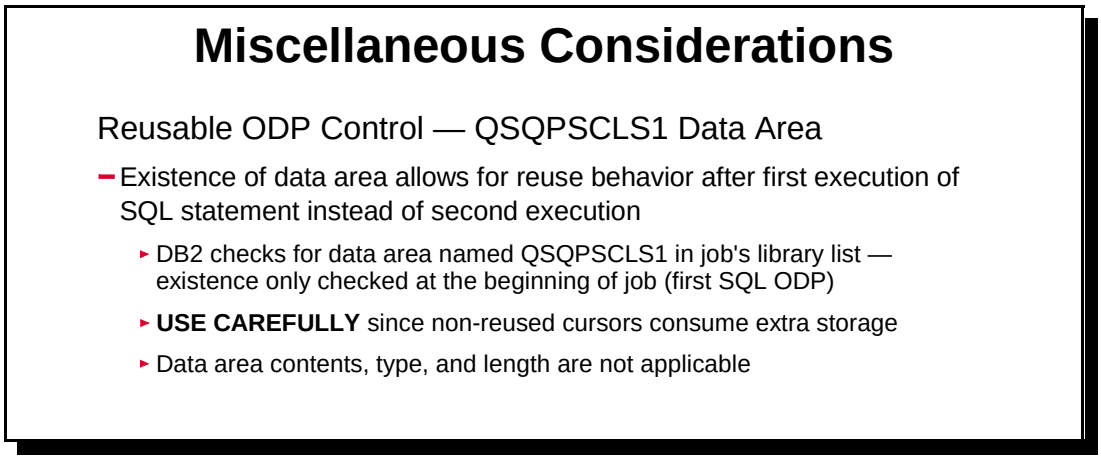
SQL Naming (*SQL) does not have the same performance overhead resource hit, since SQL Naming (*SQL) on DB2 UDB for iSeries looks for tables in the library that have the same name as those in the user profile.

Please take a few extra moments to thoroughly understand this chart before going on to the next chart.




## Miscellaneous Considerations — Data Types

Good coding practice for DB2 UDB for iSeries requires that you do NOT use compatible data types between the local host variables and table columns, as this causes the system to do unnecessary conversion and extra work. The “DB2 UDB for iSeries SQL Programming” guide has a data type equivalency section for each host language. (See the examples shown at the bottom of this chart.) Please refer to this guide for the most efficient ways of using compatible data in DB2 UDB for iSeries. The Web site for this book is found in the Links section of this course.




## Miscellaneous Considerations — Reusable ODP Control

In the “SQL Performance Basics” online course, we pointed out that you can tune DB2 UDB for iSeries so that reusable Open Data Path (ODP) processing can start after the first execution of an SQL statement, instead of waiting until the second execution. The QSQPSCLS1 data area is the interface for performing this tuning of DB2 UDB for iSeries.

Existence of the QSQPSCLS1 data area in the job's library list allows the reuse of an SQL statement’s ODP after the first execution of the SQL statement. This is in lieu of the need to wait for the second execution (default behavior) to establish a pattern of ODP reuse.

During the first execution of an SQL statement, DB2 UDB for iSeries only checks for the existence of the QSQPSCLS1 data area at the beginning of the job. DB2 UDB for iSeries will search through the job’s library list to find this data area.

Remember, use the reusable ODP controls carefully, since ODPs that are left open for reuse will consume extra storage in DB2 UDB for iSeries. Data area contents, type, and length are not applicable when creating the data area; the user can pick any values.

## 1st Mini Quiz

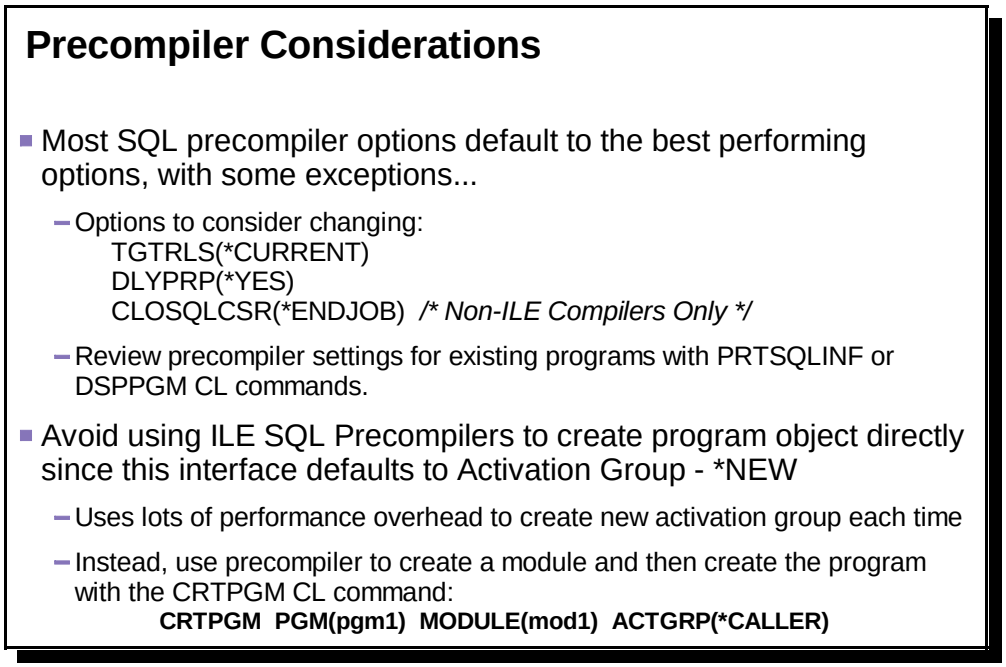
1. DB2 UDB for iSeries will only perform blocking when the application has specified a blocked SQL request.
   - ❑ True
   - ☑ False

2. SQL blocking of fetches can be improved most efficiently when you mix single and multiple row FETCH requests on the same cursor.

   - ❑ True
   - ☑ False

3. When considering the use of VARCHAR columns in DB2 UDB for iSeries, if your primary goal is improving performance rather than saving memory, you'll want:

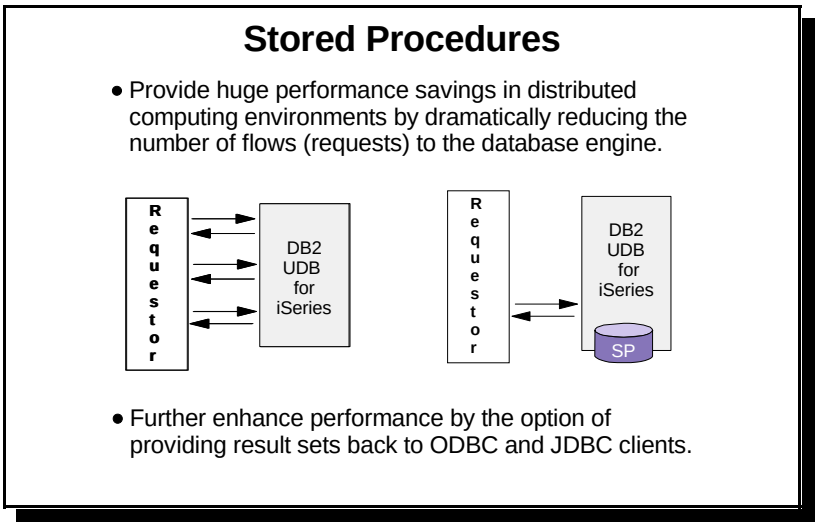
   - ❑ To set the ALLOCATE setting for a VARCHAR column to 0.
   - ☑ The ALLOCATE value to accommodate 90-95 percent of the variable length values.
   - ❑ The ALLOCATE value set equal to the VARGRAPHIC for primary key columns.
   - ❑ All of the above.




## Precompiler Considerations

The majority of the SQL precompiler options default to the best performing choices. You can use the Print SQL Information (PRTSQLINF), Display Program (DSPPGM) or Display Service Program (DSPSRVPGM) CL commands to review options currently in place for existing programs.

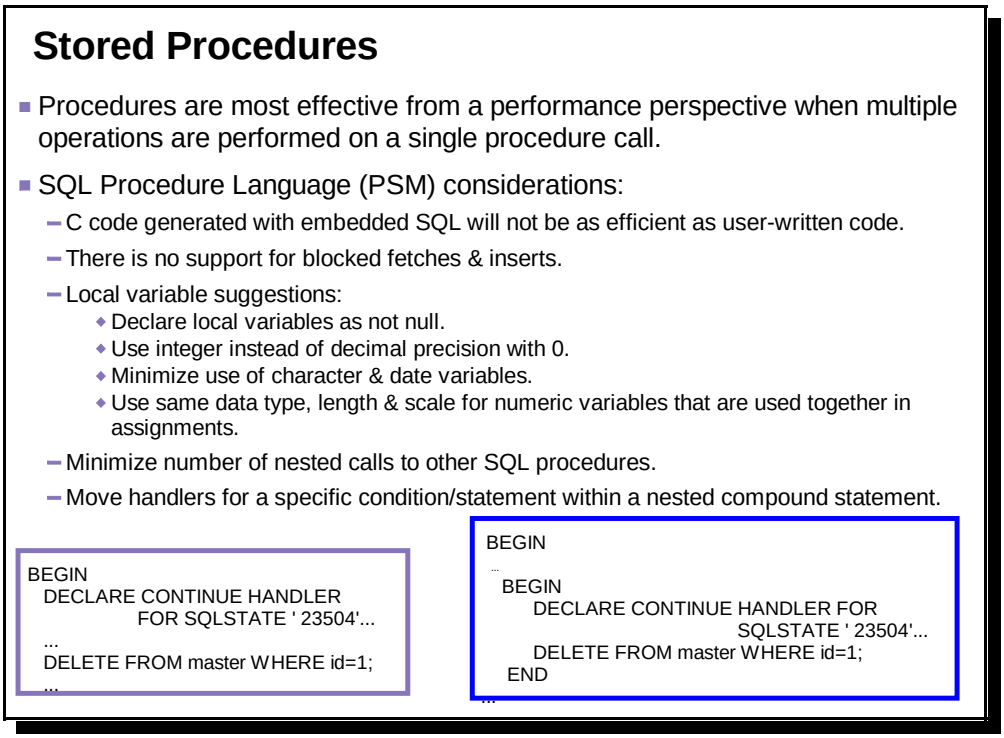
If you use any of the ILE SQL precompilers to create a program object, then that program is always created with the Activation Group attribute set to " *NEW." The *NEW attribute is not the best option since it demands extra performance overhead by creating a new activation group each time the program is invoked. Therefore, you should use the precompiler to create a module object (*MODULE), and then use the Create Program (CRTPGM) CL command to create the program object. Using the module object will give you a better performing activation group object. See the CRTPGM CL command sample shown at the bottom of the chart.




## Stored Procedures

Substantial reduction of database engine resources can be obtained in distributed computing environments by dramatically reducing the number of requests sent to the engine. Stored procedures contribute to efficient SQL performance in distributed types of environments. The benefit of stored procedures is that they allow the same amount of database work to be performed with fewer trips to the database server by bundling together DB2 UDB for iSeries requests into a stored procedure. Additional performance improvements can be achieved by using the option of providing result sets back to ODBC and JDBC clients.

For more information on Stored Procedures, see the Redbook, *Stored Procedures, Triggers, and User-Defined Functions for DB2 Universal Database for iSeries*. The Web site for this book is found in the Links section of this course.

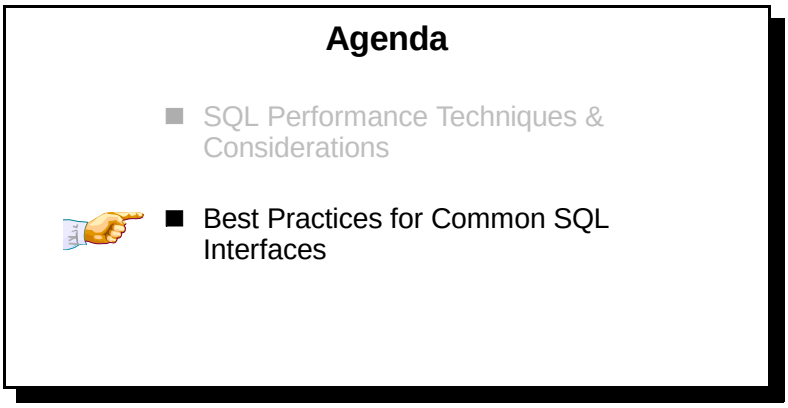



## Stored Procedures

Stored procedure calls are most effective at improving performance when the procedure performs multiple operations. It's not recommended that a stored procedure performs a single SQL statement or task because of the overhead involved with calling the stored procedure.

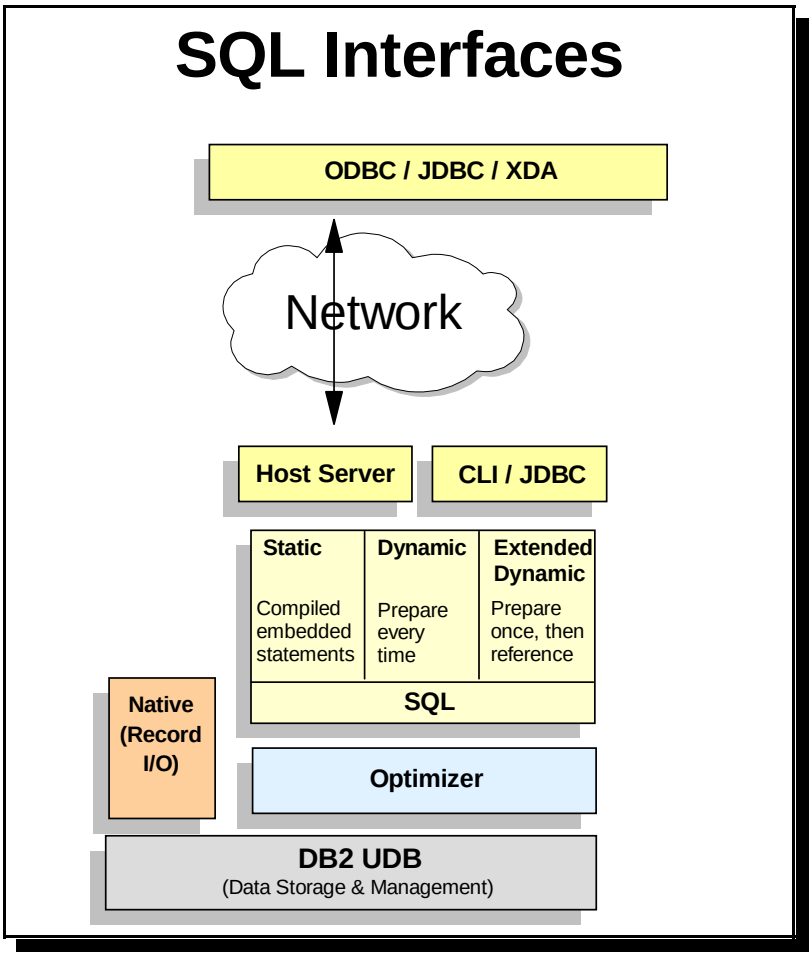
SQL stored procedures (and functions and triggers) are implemented on DB2 UDB for iSeries by generating C program objects. Since generated code is almost never as efficient as handwritten code, programmers should utilize the tips documented on this chart and in the “V5R3 DB2 UDB for iSeries SQL Programmer's Guide” to get the most efficient code generation out of the database engine. (The Web site for this guide is found in the Links section of this course.)

This completes the section on SQL programming techniques for high-performance. Our agenda continues by examining the best practices for implementing common SQL interfaces.




## Agenda: Best Practices for Common SQL Interfaces

We'll spend the second half of this course by examining those coding techniques that represent the best practices for common SQL interfaces for DB2 UDB for iSeries.




## SQL Interfaces

First, let's take a high-level look at the DB2 UDB for iSeries architecture and the paths that an SQL request can take. You've learned all of the general SQL performance techniques in previous sections. Now, we'll concentrate on detailed techniques for specific SQL interfaces. We will, of course, pay special attention to the Call Level Interface (CLI) and JDBC server query program interfaces that support host-based queries, as well as ODBC, JDBC, and XDA (eXtended Dynamic SQL APIs) ) client query program interfaces that support network clients.

## Common SQL Application Interfaces

| | CLIENT-based | SERVER-based |
|---|---|---|
| **ODBC** | iSeries Access ODBC Driver - *includes OLE DB & .NET Provider* | DB2 UDB for iSeries Call Level Interface (CLI) |
| **JDBC** | iSeries Java Toolbox JDBC Driver | iSeries Native JDBC driver - *including SQLJ support* |

## Common SQL Application Interfaces

SQL application interfaces come in two flavors: those that are client-based and are designed to serve the needs of client or distributed queries, and those that are server-based. Although there are third-party ODBC and JDBC drivers, this section of our course will focus on concepts related to the standard IBM drivers that are provided at no additional cost for iSeries servers. However, some of the concepts presented here will also apply to third-party drivers.

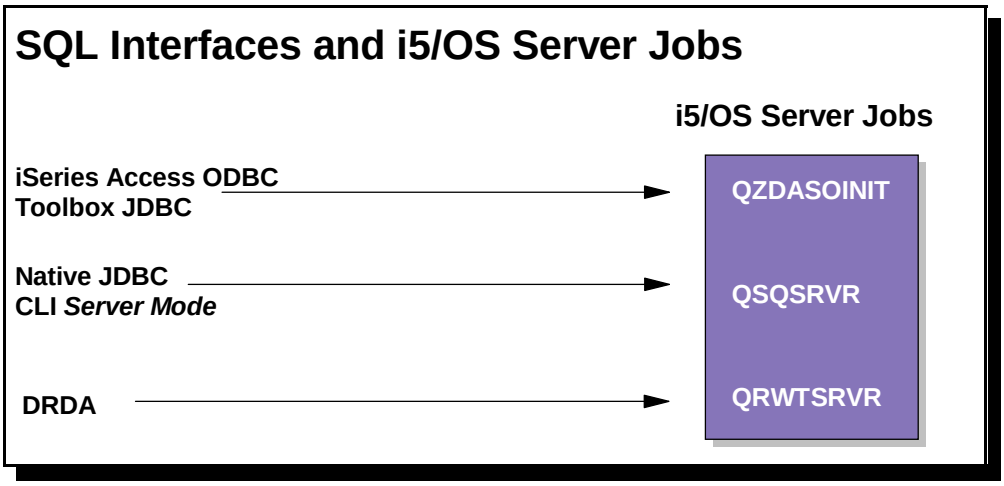
A key difference in the two types of interfaces is that the client-based drivers, ODBC and JDBC, have interface customizations, such as extended dynamic, that do not necessarily exist in the "server-based" interfaces.

SQL Call level interface (CLI) and the iSeries native JDBC driver (which is built on top of CLI) are no-charge features for DB2 UDB for iSeries and are shipped with every iSeries server. The native JDBC driver is actually part of the iSeries Developer Kit for Java™ and is a Type-2 JDBC driver.

The iSeries Access ODBC driver and IBM Toolbox for Java JDBC driver are part of the iSeries Access for Windows offering. In addition to the ODBC driver, this package also includes an OLE DB and Microsoft® .NET provider. The IBM Toolbox for Java JDBC driver is a Type-4 JDBC driver and also known as the JTOpen JDBC driver.

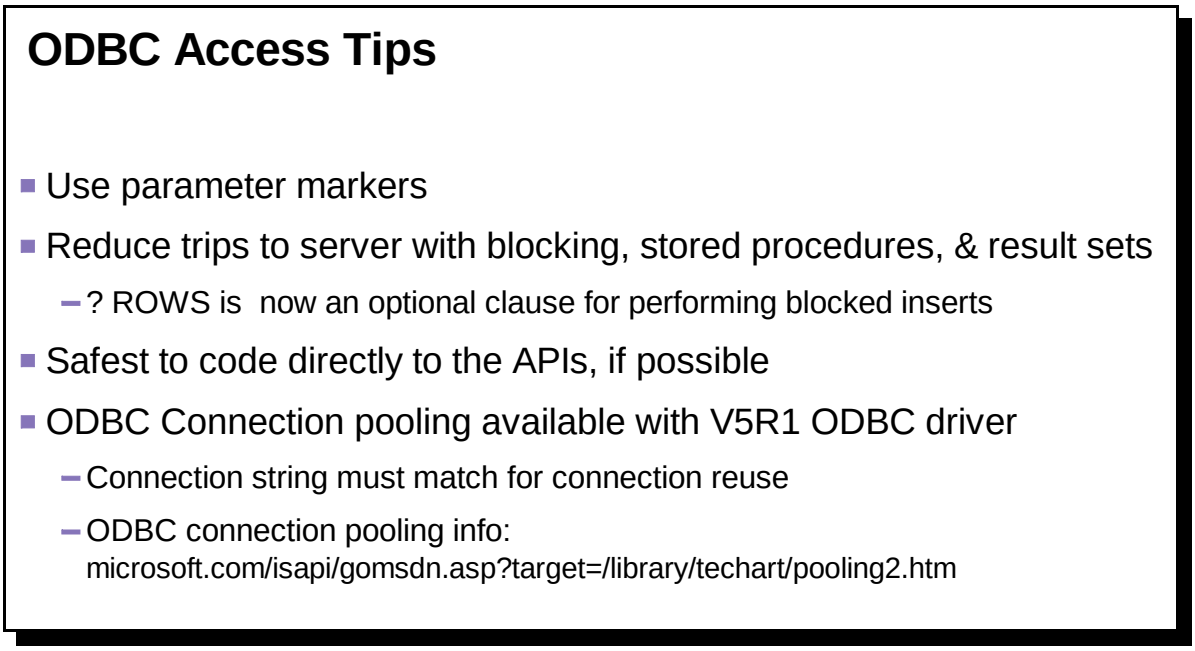
Notice that the tuning of both the ODBC driver and the IBM Toolbox for Java JDBC driver are similar since they are each client-based interfaces. The same holds for the two server-based database interfaces: CLI and the iSeries native JDBC driver.

For FAQs related to the iSeries Developer Kit for Java JDBC, go to the Web site listed in the Links section of this course.




## SQL Interfaces and i5/OS Server Jobs

If you're trying to monitor or trace the work being done by one of these SQL interfaces, this graphic shows which i5/OS™ and OS/400® server jobs are used by each of the different interfaces.




## ODBC Access Tips

Here are some tips on the best ways to access ODBC—which is a dynamic SQL interface. Your use of parameter markers is very important.

Stored procedures, result sets, and blocking are also key to remote clients that are performing database requests. This is because they reduce the number of trips required to the database server.

Here's advice worth heeding—avoid the Microsoft Jet Database Engine since it can generate very inefficient ODBC requests against any database server, not just iSeries servers. That inefficiency can be wisely avoided by coding to the ODBC APIs instead.

Pooling of the ODBC connection objects on the requestor side is another way to improve the performance of ODBC-based applications. ODBC connection pooling FAQ information is available at the Web site provided in the Links section of this course.

# ODBC Access Tips

## Extended Dynamic Access

- Use the extended dynamic feature of Client Access ODBC driver (also available to IBM Toolbox for Java JDBC driver)
  - Provided as a feature which is externalized as data source option
  - Allows for private or shared packages
  - Reduces line trips by up to 33% when used in conjunction with "Cache Package Locally"
    - At connection time, portions (statement text, result & parameter marker descriptions) of the extended dynamic package are downloaded.
    - Many ODBC "setup" APIs can use the descriptive info cached on the client instead of going up to the server.

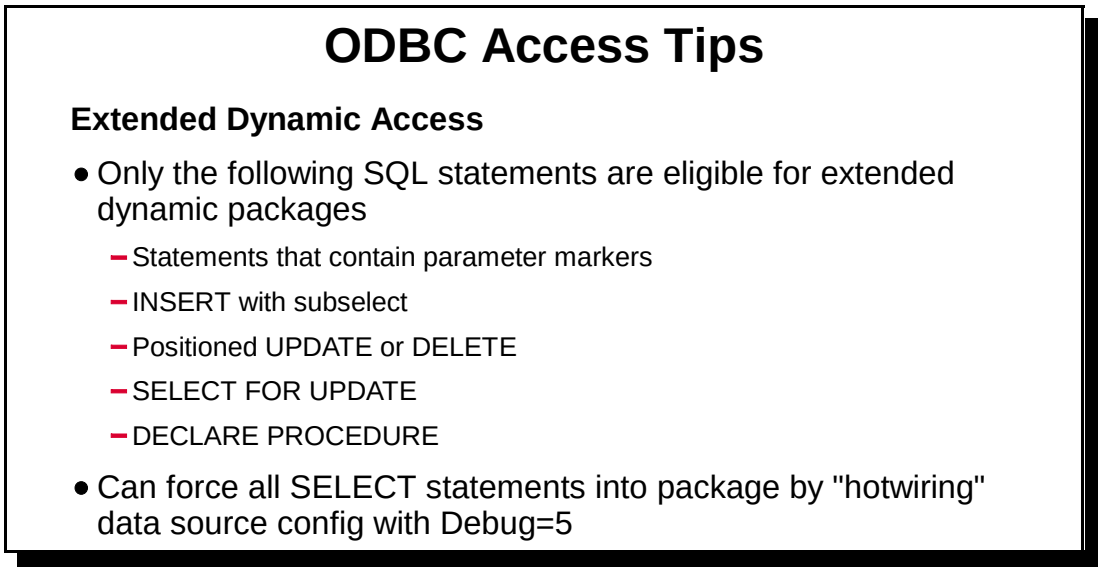
## ODBC Access Tips — Extended Dynamic Access

The Extended Dynamic Access feature has been available with the iSeries Client Access ODBC driver for a while, but it's much more usable since the release of the larger .5 gigabyte package size limit in V4R3. The Extended Dynamic Access feature permits the use of private or shared packages. (The Extended Dynamic Access feature is also available to the iSeries JDBC driver.)

The Extended Dynamic Access feature, when used in conjunction with the Cache Package option, can reduce line trips to the server by one third. It does so by storing information on the parameter markers and setting the result locally on the client for an SQL statement. At connection time, parts of the extended dynamic package are cached down to the client. Specifically, the statement text, parameter marker, and result set descriptions are copied down to the client. Some setup ODBC APIs must be executed to get parameter markers and result set areas "ready" for execution. However, when the marker and result set setup is performed, a trip to the server is not required. Rather, the information is retrieved from the client cache. This improves overall performance.

Private packages can provide some advantages if various users are executing different statements. But, shared packages can cause problems when large numbers of new statements are being added, because the packages must be locked while statements and access plans are added.

There is no special switch for private versus shared packages. Instead, for shared packages, you must configure each shared user to supply the same application name, same package name, and library. Then, check the "Use" package option for each shared user. The users' ODBC data sources also need to have the same default library value.




## ODBC Access Tips — Extended Dynamic Access (continued)

This chart continues our examination of ODBC Extended Dynamic support.

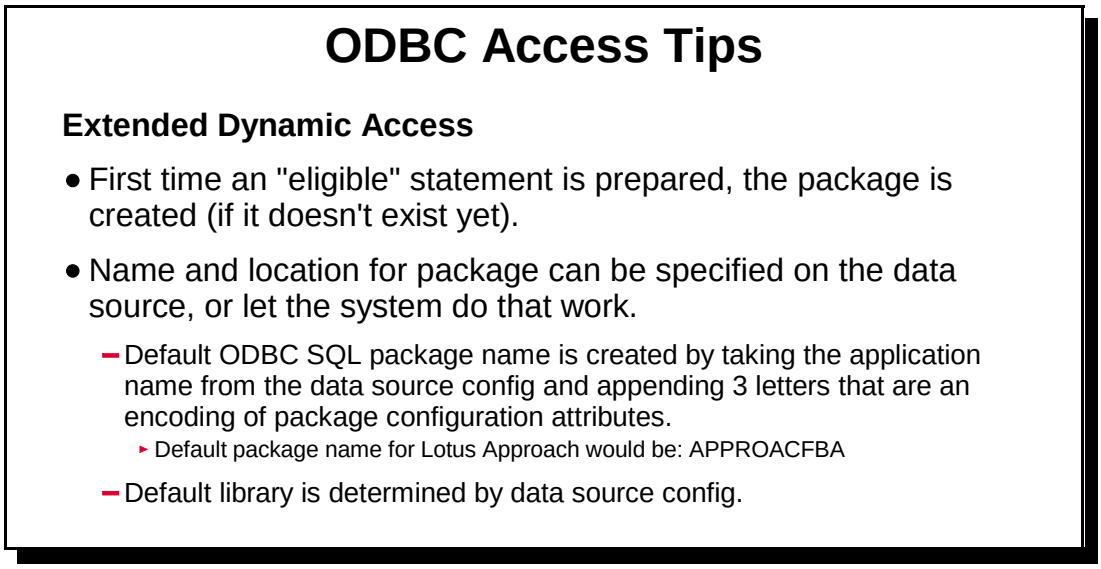
It's important to know that only the SQL statements shown on this chart are eligible for ODBC extended dynamic packages. These include:

- Statements that contain parameter markers
- The INSERT statement with Subselect
- The positioned UPDATE or DELETE statements
- The SELECT FOR UPDATE statement
- The DECLARE PROCEDURE statement

Unlike the system SQL Process Extended Dynamic (QSQPRCED) API, a limited number of preparable SQL statements can be saved in the package. (The QSQPRCED API was reviewed in the SQL Performance Basics online course that is a prerequisite to this course.)

If you want to increase the types of statements that can be placed into the package, the ODBC data source can be hotwired to force all SELECT statements into the package. (We'll review "hotwiring" instructions later in this section).

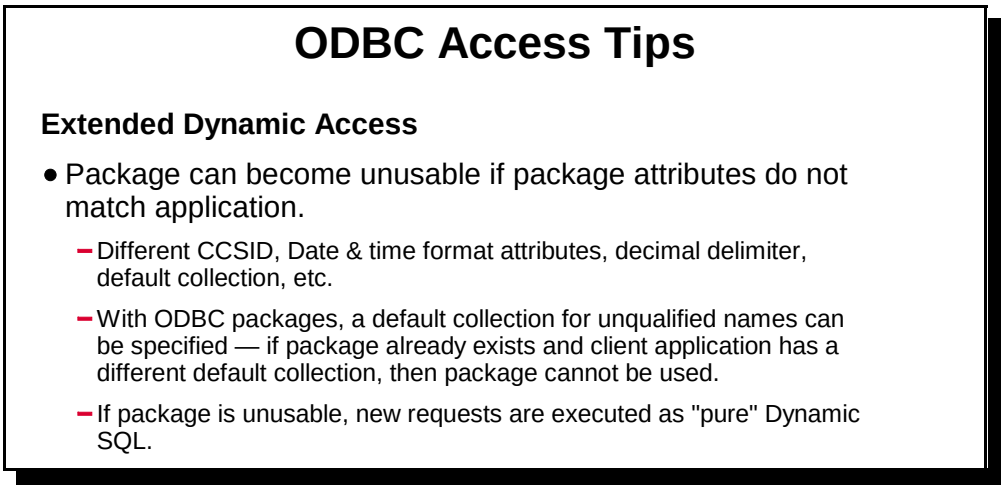
Automatic parameter marker conversion does NOT happen on ODBC access, because DB2 UDB for iSeries only allows SELECT statements with parameter markers into the package. That system feature is disabled on the ODBC and JDBC (IBM Toolbox for Java) interfaces.




## ODBC Access Tips — Extended Dynamic Access (continued)

The ODBC/JDBC Extended Dynamic support will automatically create the package when an eligible statement is prepared, if the package doesn't already exist.

If you don't want the system default behavior, you can specify your own package name and library. The default system naming behavior takes the first seven characters of the data source application name, then appends three letters that are an encoding of the package configuration attributes. For example, the default package name for Lotus® Approach would be: APPROACFBA. The default library is determined by the data source configuration.

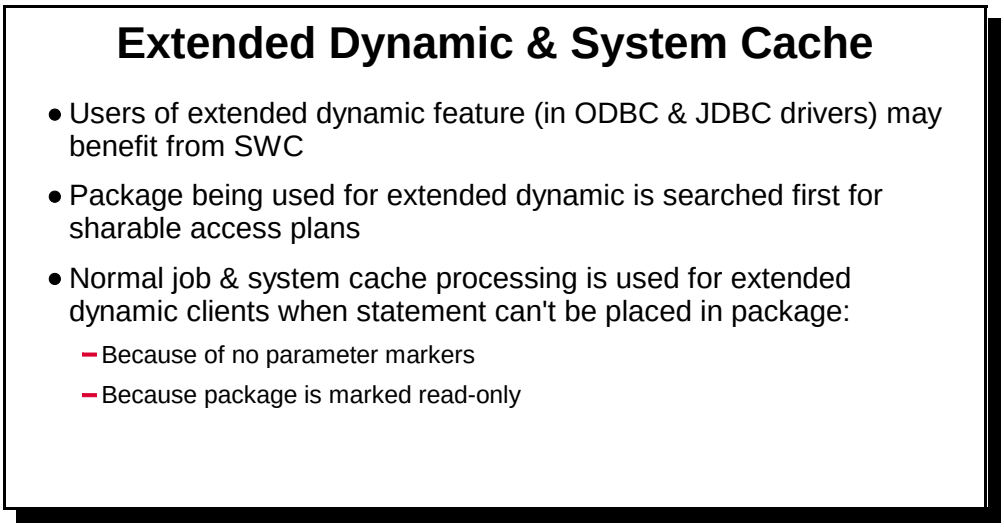



## ODBC Access Tips — Extended Dynamic Access (continued)

The extended dynamic package can become unusable if its attributes don't match the application. You can determine if the "unusable" condition has occurred by looking into the job log for the message "Extended Dynamic has been disabled" to determine if ODBC was unable to use an SQL package because of differences in CCSID, date and time format attributes, decimal delimiter, default collection, etc.

With ODBC packages, a default collection for unqualified names can be specified. But, if that package already exists and the client application has a different default collection, then the package cannot be used.

If an ODBC Extended Dynamic package becomes full, then the driver detects the package's full condition (SQL0904, reason code 7), and switches over to dynamic SQL for newly prepared statements. Switching from extended dynamic to dynamic SQL has some negative performance consequences. However, statements that were previously prepared in the SQL package will continue to benefit from the package.




## Extended Dynamic & System Cache

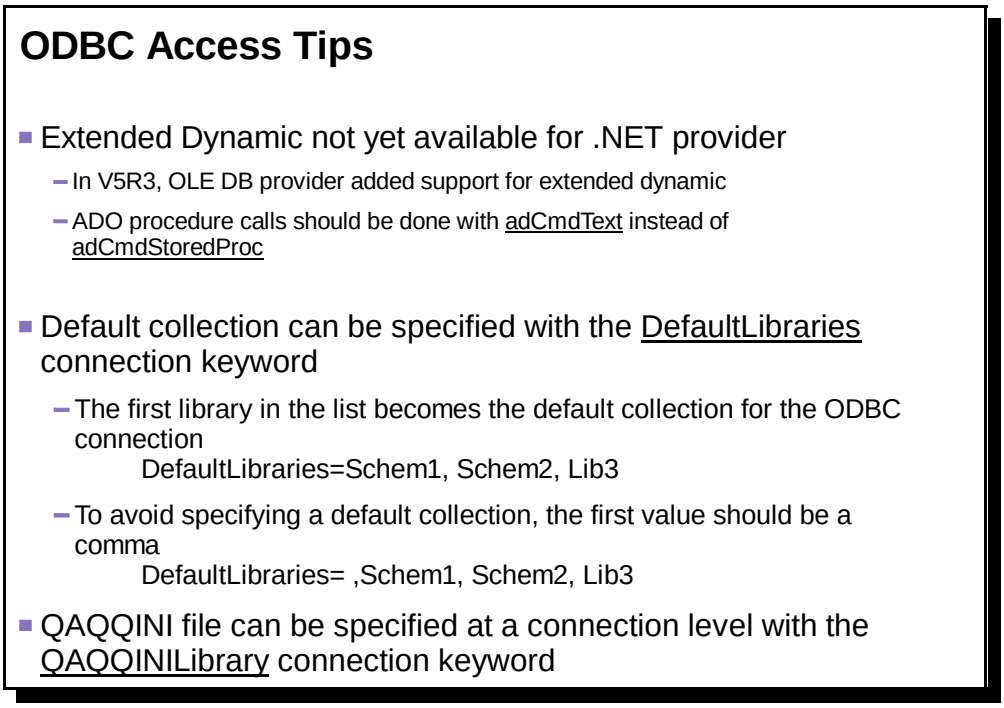
Here's a summary of how extended dynamic packages and the System-wide Statement Cache (SWC) interact.

Users of the extended dynamic feature that is available with the ODBC and JDBC drivers may benefit from the Systemwide Statement Cache (SWC).

The package being used for extended dynamic is always searched first for access plans that can be shared.

The normal Job and System cache processing will be utilized for extended dynamic clients when the statement cannot be placed into the package ...

- It doesn't use parameter markers.
- The package has been marked "read only."

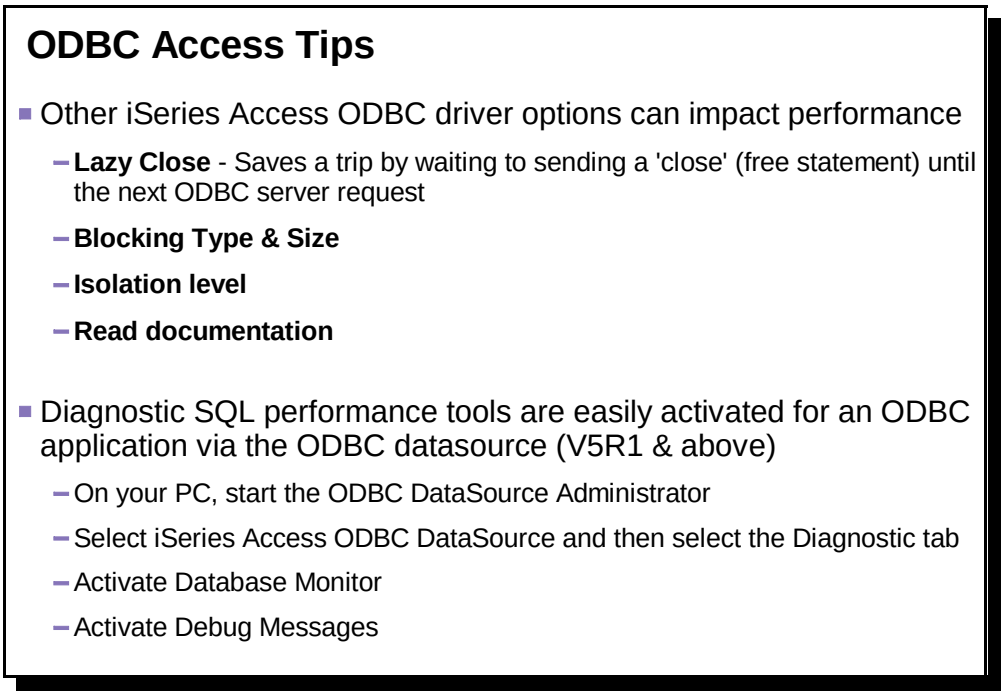



## ODBC Access Tips

The Extended Dynamic option is not available when using the V5R3 .NET provider data access middleware. The V5R3 OLE DB provider does support the Extended Dynamic feature.

As discussed in the "SQL Performance Basics" course, setting the default collection can improve performance in some cases. This chart shows examples of how the default collection can be specified with ODBC Connection keywords.

The QAQQINI options file can be used to override some of the default DB2 UDB for iSeries settings. The ODBC driver supports a connection keyword to make it easier to activate a query options (QAQQINI) file for a connection. More information on QAQQINI settings can be found online at the Web site listed in the Links section of this course.




## ODBC Access Tips

This chart lists the other ODBC data source options that can have an impact on system performance. They are:

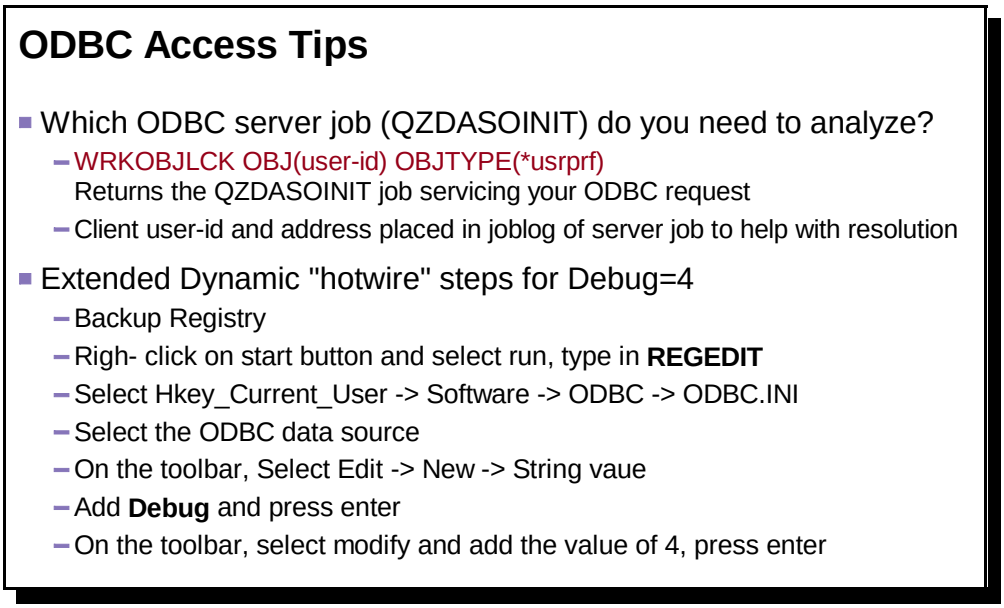
- Lazy Close — This option saves a trip to the server by waiting to send a close (which is a free statement) until the next ODBC server request is received.

- Blocking Type and Size — This option takes advantage of the inherent efficiencies offered by blocking. You can play with various block sizes to determine the optimum for various situations.

- Commitment control level — As mentioned earlier, commitment control can impact performance by the number and duration of locks obtained by DB2 UDB for iSeries. Examine your application concurrency requirements carefully before choosing your commitment control level.

The iSeries Access for Windows documentation contains good information on these configuration options. The Web site for this information is located in the Links section of this course.

Starting with V5R1, the ODBC driver has also made it easier to activate the database monitor and optimizer debug message traces for ODBC connections that need to be analyzed for performance.

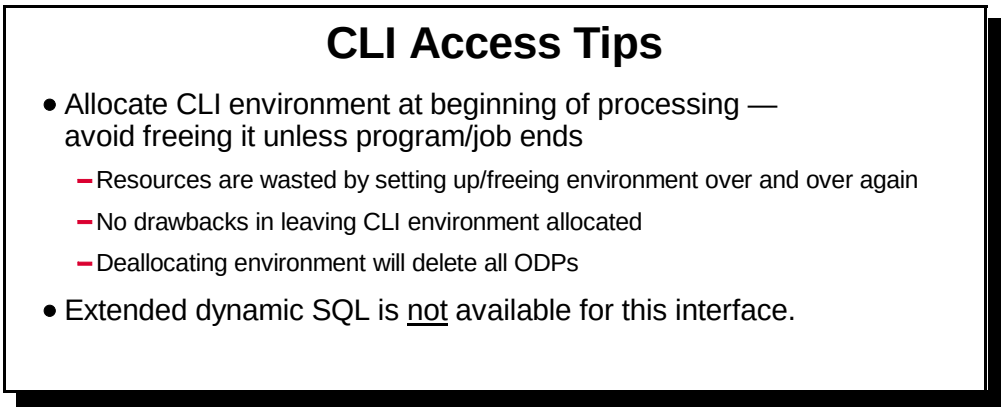



## ODBC Access Tips

If you've ever attempted to determine which ODBC server job ( QZDASOINIT) is servicing your client so that you can either turn on optimizer debug messages or start the database monitor, then you know it is a complex problem. This Work Object Lock (WRKOBJLCK) tip makes analyzing it a little easier, assuming each ODBC connection is using a unique userid.

- From your console, enter the WRKOBJCK command shown in red on the chart:
    - This will return the QZDASOINIT job that is servicing your ODBC request.
    - The client userid and address are placed into the server joblog to help with the resolution.

Shown on this chart are the “hotwiring” steps we promised earlier in this course for Extended Dynamic when Debug equals four. It involves using REGEDIT. First, backing up your registry is good protection. You may want to print out this chart so you can refer to these steps later.

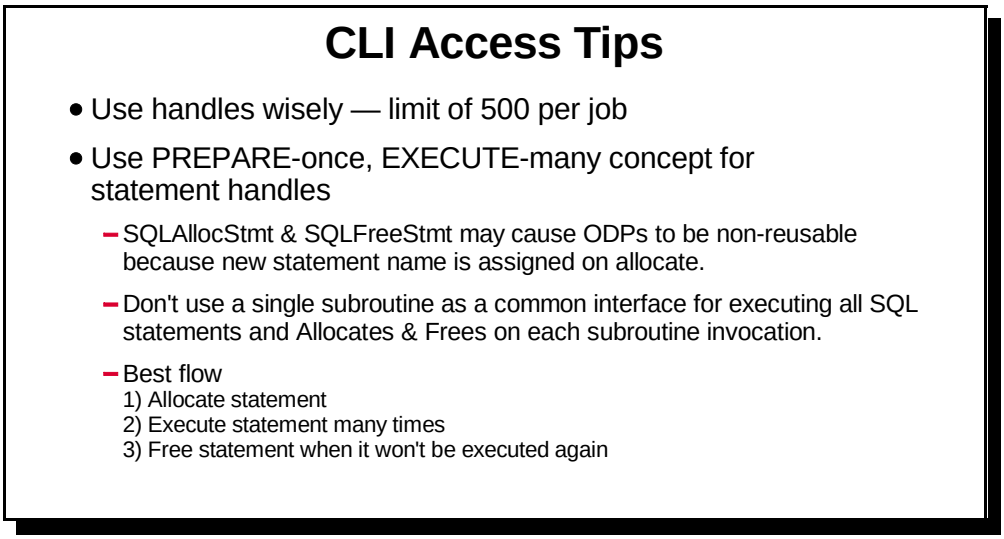



## CLI Access Tips

The server-side equivalent of ODBC is CLI. Although the APIs used by these two interfaces are almost identical, their performance personality on iSeries servers is quite different.

Here's a tip: The best way to allocate your CLI environment is to do so at the very beginning of processing. Then, avoid freeing it unless the application (or job) is actually ending. The reason for this is that there is some setup done when the CLI environment is allocated, and that setup must be cleaned up when the environment is freed. If the job will use the same process many times, resources are wasted by continually setting up and freeing the environment. The biggest resource waste occurs when all ODPs are deleted as the CLI environment is deallocated. There are no disadvantages to just letting the CLI environment remain active until you actually end the job.

[*NOTE: CLI doesn't provide any support for SQL extended dynamic.]*




## CLI Access Tips

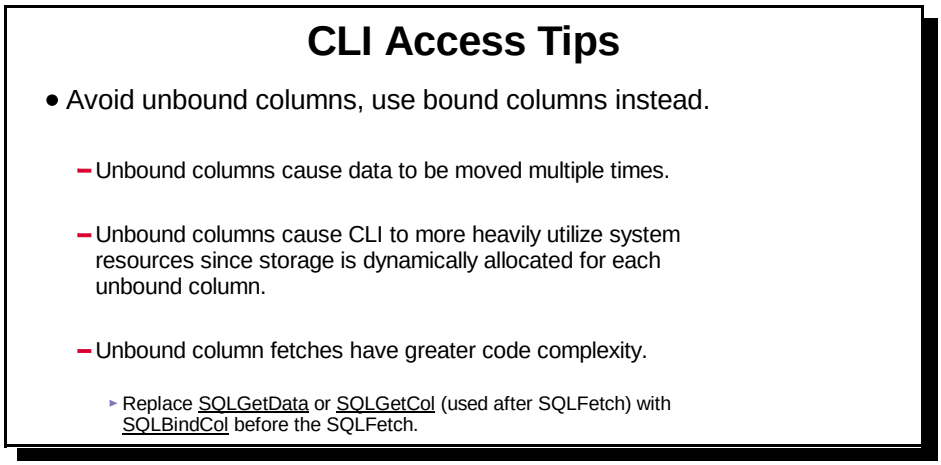
Avoid running out of statement handles, because the only way to get more handles after the limit of 500 is reached is to reallocate the CLI environment. As was covered on the preceding chart, reallocating the CLI environment causes the deletion of all active ODPs, so you don’t want to do this.

The Allocate and Free statements use a new statement name each time they are executed. Therefore, you will not receive the benefit of ODP reuse that can occur each time the database performs the same Dynamic SQL. This is because the statement name will never match any of the cached statements.

CLI requests should use the best flow algorithm shown on this chart to get the best performance. That is, they should:

- Allocate the statement.
- Execute the statement many times to conserve resource.
- Free the statement when the program or job finishes and won’t again be executed.

It is possible to increase the handle limit with the CLI Server Mode that prompts your CLI program to run in a threaded fashion. But, moving an application to a threaded model requires very careful consideration, and is beyond the scope of this course.

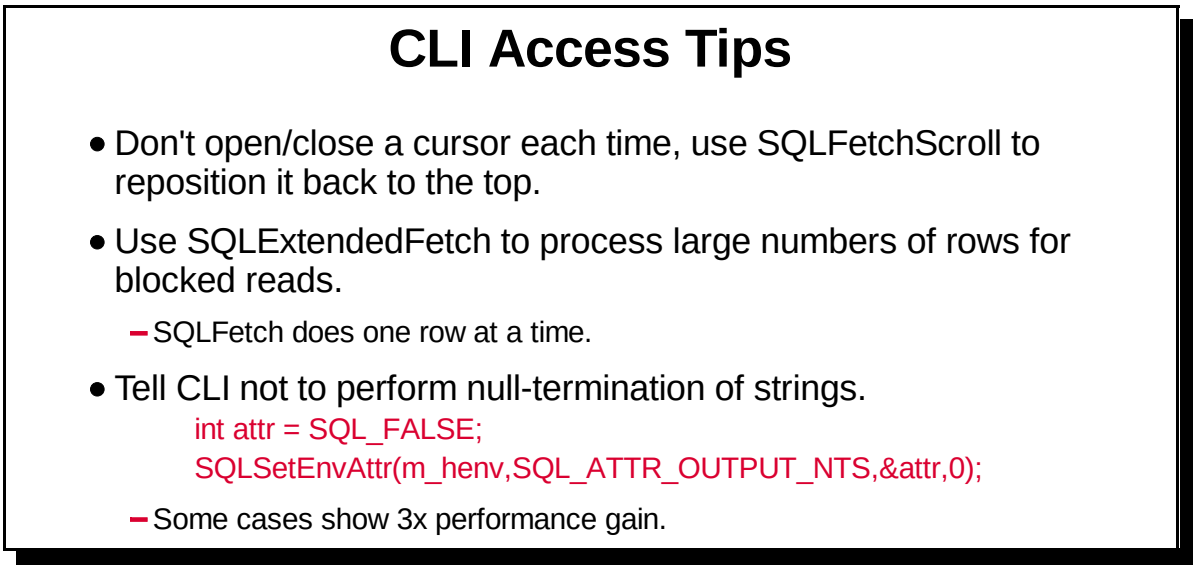



## CLI Access Tips

Let's continue with additional CLI Access tips.

With unbound columns, the system has to dynamically allocate storage. This is because it has no prior data about the columns that will be stored. (The SQLBindCol does let DB2 UDB for iSeries "prepare" for the columns ahead of time, but we'll talk about this more in just a minute). The dynamic nature of unbound columns usually causes the data to be moved multiple times, which is inefficient. So, you should avoid using unbound columns whenever possible. Instead, use bound columns for greater resource efficiency:

- Unbound columns require your data to be moved multiple times.
- Unbound columns require CLI to use more system resource, because storage is dynamically allocated for each unbound column. This may result in resource exhaustion, with storage consumption making none available for the next CLI function.
- Unbound column fetches require more complex code than do bound columns. So, for example, you should locate any code where SQLGetData or SQLGetCol is used after SQLFetch, and replace the code with SQLBindCol before the SQLFetch.




## CLI Access Tips

Instead of opening and closing the cursor each time, use the SQLFetchScroll function to reposition the cursor back to the first row of its result set.

Make use of SQLExtendedFetch when processing large numbers of rows to benefit from the efficiencies of blocking. SQLFetch can only process one row at a time, and blocking is not available with SQLFetch.

The null-termination of strings option should not be used when a single general purpose buffer is being utilized as the destination for fetch operations from different tables—this forces the computation of the data length each time.

## CLI Access Tips

- New statement attribute forces a new ODP to be created each time the statement is executed.

```
long attr;

attr = SQL_TRUE;

SQLSetStmtAttr(hstmt, SQL_ATTR_FULL_OPEN, &attr, 0);

SQLExecute(hstmt);
```

### CLI Access Tips

In a few rare cases, forcing the creation and deletion of an ODP can be good for query performance. This example shows how to force this behavior with the CLI Full Open statement attribute.

## CLI Access Tips

- Unless you have an explicit need to update, set all cursors to be read only

  attr = SQL_TRUE;

  SQLSetStmtAttr(hstmt,SQL_ATTR_FOR_FETCH_ONLY,&attr,0);

  where hstmt is the statement handle you want to make read-only

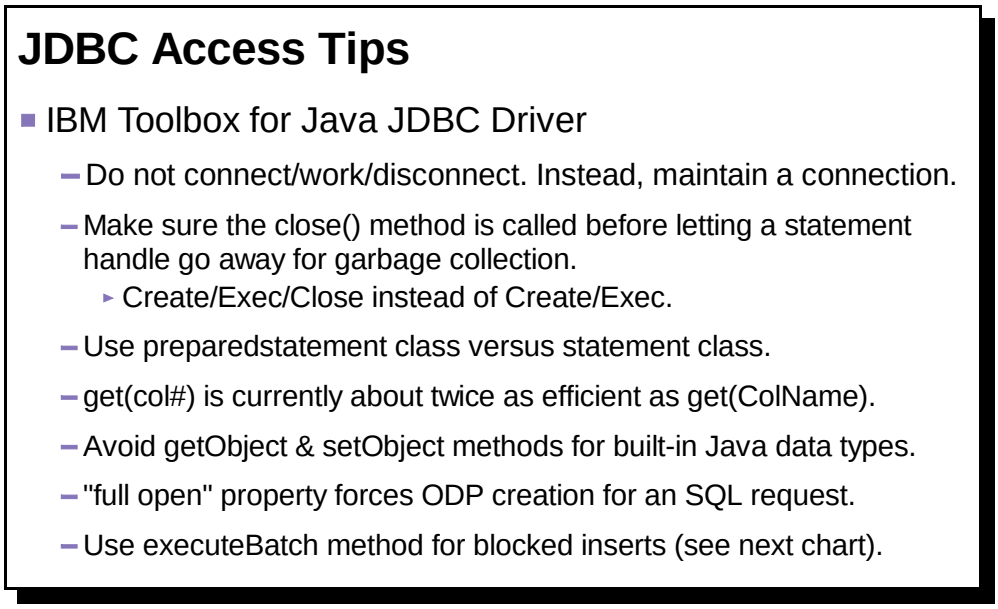
- Limit use of catalog functions (SQLTables, SQLcolumns, etc.) & provide very narrow search criteria on catalog request—since catalogs tend to be very large.
- CLI FAQ at http://www.iseries.ibm.com/db2/clifaq.htm

### CLI Access Tips

The implementation of CLI on iSeries does tend to default more to update-capable cursors than is the case with other DB2 implementations. So, you should take the time to intentionally set all cursors to be read only, unless you have an explicit need for update-capable cursors in your application. This chart shows the steps required for making a CLI cursor read only. (See the code shown in red.)

You can conserve system resources by limiting the use of catalog functions, such as SQLTables and SQLColumns. You can also provide very narrow search criteria on catalog requests, because catalogs tend to be large.

Many other CLI tips and questions are contained in the online SQL CLI FAQs, located at the Web site shown on this chart and also listed in the Links section of this course.




## JDBC Access Tips

The "client" JDBC driver is packaged with the IBM Toolbox for Java. The "client" JDBC driver has many of the same performance considerations and tuning characteristics as the Client Access ODBC driver. For example, a JDBC connection property can be set to enable extended dynamic access on this interface, as is true with the ODBC driver.

The *preparedstatement* class is more efficient, and has a better chance for reusable ODPs than does the statement class. So, you should use this whenever possible.

*get(col#)* is faster than *get(colname)* because the *get(colname)* function has to perform a lookup with the column name to get the column number. That lookup step is bypassed when using *get(col#)*, making it approximately twice as efficient as *get(colname)*.

Keep in mind that you should not use the traditional logic flow of “connect, work, then disconnect.” Instead, maintain a connection — everything will work more efficiently if you keep this goal in mind.

Make sure the *close()* method is called before you delete a statement handle. This can be accomplished by using *Create/Exec/Close* rather than simply *Create/Exec*.

Extended Dynamic can be enabled in the IBM Toolbox for Java JDBC driver by setting the “Extended Dynamic” connection property.

As with SQL CLI, the IBM Toolbox for Java JDBC driver supports a "full open" statement property for those rare cases where forcing the creation of an ODP can improve query performance.

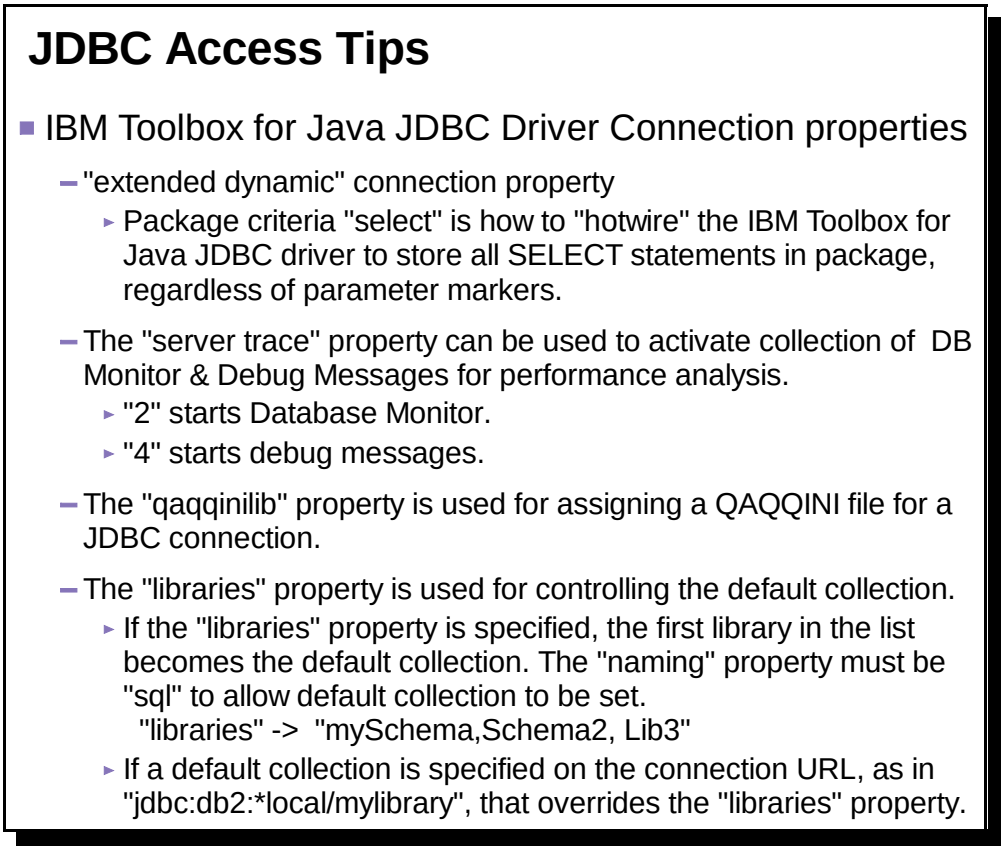
## JDBC Access Tips

- IBM Toolbox for Java JDBC Driver - Blocked Insert Example

```
PreparedStatement pstmt =
        con.prepareStatement("INSERT INTO teams VALUES( ?, ?, ?, ?,)");

for (int j = 0; j < numOfLoops; j++) {
  pstmt.setString(1, "Team_" + Integer.toString(j));
  pstmt.setInt(2, i);
  pstmt.setFloat(3, 4.99f);
  pstmt.setInt(4, 0);
  pstmt.addBatch();

 }

int [] updateCounts = pstmt.executeBatch();
```

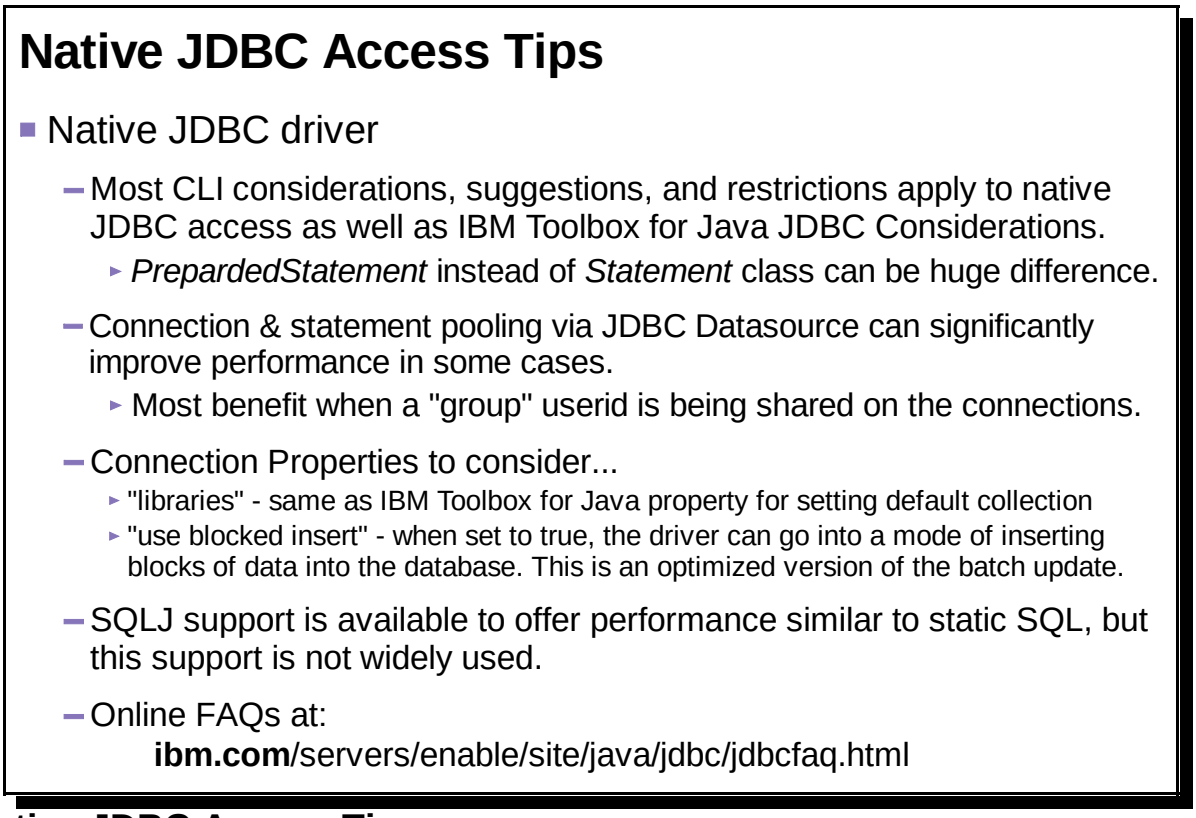
## JDBC Access Tips

The performance benefits of blocked SQL requests were discussed earlier. Here you can see how to perform a blocked insert with JDBC by studying the code shown in red.




## JDBC Access Tips

The performance benefits of these connection properties have been discussed previously. This chart shows how these connection attributes are used from a JDBC interface.




## Native JDBC Access Tips

The iSeries native JDBC driver is built on top of CLI. Therefore, all the same CLI considerations and suggestions apply. In addition, almost all of the IBM Toolbox for Java JDBC driver considerations also apply.

The *PreparedStatement* class is much more resource-efficient than the *Statement* class for both JDBC drivers. Tests run in the lab to compare the efficiency of these two classes on a statement that is run 20 times yielded the following results:

- For the statement class, the driver called the database 120 times.
- For the preparedstatement class, the driver called the database six times (a 95% savings!!!)

Unlike the IBM Toolbox for Java JDBC driver, the Native JDBC driver does support standard classes for pooling of connections and statements. JDBC application performance can be improved substantially with this pooling technique. However, the Native JDBC driver does require an extra connection property, *use blocked insert*, in order for the *ExecuteBatch* method to generate a blocked insert statement.

SQLJ support is available with the Native JDBC driver, but it's not been widely used since the pre-processing steps are not portable across JVMs.

For FAQs on the iSeries Developer Kit for Java JDBC, go to the Web site listed in the Links section of this course.

## Java Data Access Considerations

- Boosting performance with Unicode...
  - If data access occurs primarily via Java applications, big performance gains (up to 60%) can be seen by...
    - Using Unicode type instead of character types to avoid translation overhead
  - Considerations when switching from character to Unicode
    - Disk: each character now requires 2 bytes instead of 1.
    - Performance: If data must be shared, then legacy applications will suffer the translation penalty.
  - Easy to switch existing character column:
    **ALTER TABLE mytable ALTER COLUMN mycol**
    **SET DATA TYPE GRAPHIC (10) CCSID 13488**
  - Performance also can be improved by explicitly converting character data on fetches:
    **SELECT VARGRAPHIC(mycol,10,13488)...** instead of **SELECT mycol ...**
  - UTF-8 & UTF-16 encodings are available with V5R3
    - UTF-8 maps to CCSID 1208
    - UTF-16 maps to CCSID 1200

## Java Data Access Considerations

Since Java always processes character data in Unicode, the best performance for JDBC can be achieved by actually storing the character data in the Unicode CCSID of 13488. This practice provides better system performance, because no character translation is required on the input and output effort of character data between Java and the underlying EBCDIC encoding. However, there are some disk storage and legacy application performance considerations. So, you'll need to consider the following two points before you make a Unicode decision:

- With Unicode, each character stored on disk will require two bytes rather than one.
- When legacy applications need to access Unicode data, they will pay a translation resource penalty.

Once you've decided to store the data in Unicode, it's easy to accomplish. See the code shown in red on this chart for an illustration of how you would do it.

Switching to Unicode has the most benefit to those applications using the Native JDBC driver. Since the IBM Toolbox for Java JDBC driver is used primarily by remote clients, Unicode character strings means that twice as much character data is sent across the network. The increase in network traffic can negate the performance savings associated with eliminating the translation from EBCDIC to Unicode. .

## Cursor Sensitivity Considerations

- Sensitivity controls whether or not the result set returned by a cursor will include changes (Insert/Update/Delete) made to a table after the cursor was opened.
  - **ASENSITIVE:** Default setting, DB2 can choose to implement the cursor as Sensitive or Insensitive. Updateable cursors are always implemented as Sensitive.
  - **SENSITIVE:** The cursor has some level of sensitivity to table changes after the cursor is opened. If changes cannot be made visible, then the Open will fail.
  - **INSENSITIVE:** The cursor is read-only and operates against a copy of the data.
- Sensitivity has a direct impact on query optimization & performance.
  - **Asensitive:** This is the best performing option because it allows the query optimizer to decide when to make copies (e.g., hash tables) of the data.
  - **Sensitive:** This forces CQE to be used and eliminates the usage of DB2 SMP and data copies/temporary results by the optimizer.
  - **Insensitive:** This forces data copies, whether it is good for performance or not.

**Cursor Sensitivity Considerations**

Cursor sensitivity refers to a cursor setting that defines whether or not the result set returned by a cursor will include recent changes to the underlying table. For example, a sensitive cursor declared as "*SELECT * FROM orders*" would return all the current orders as well as any new orders inserted into the database after the cursor was opened. For example, if there were there six orders when a sensitive cursor was opened and two new orders inserted after the cursor was opened by another application, then a sensitive cursor will try to include all eight orders in its result set. In contrast, an insensitive cursor will tend to return only the original six orders.

The default setting, *Asensitive*, means the query optimizer will choose the setting. Before changing to the *Sensitive* or *Insensitive* setting, extra consideration should be given since these options can negatively impact query optimization. The reason for this is that the non-default options limit the choices the query optimizer can make.

This chart lists possible negative impacts of the Sensitive & Insensitive settings. In general, the best performance will be obtained by specifying a value of Asensitive; thus, allowing DB2 UDB for iSeries to make the best choice for a particular SQL request.

For more details on how to control this sensitivity settings on various interface, reference the "cursor sensitivity" Web site listed in the Link section of this course.

## 2nd Mini Quiz

1. What is a good way to utilize the ODBC driver?

   - ❑ Use parameter markers
   - ❑ Use stored procedures, blocking and result sets
   - ❑ Code directly to the APIs and avoid using the MS Jet Engine
   - ☑ All of the above

2. There are no performance consequences associated with ending and restarting an ODBC or JDBC connection

   - ❑ True
   - ☑ False

3. Which of the following is true about unbound columns?

   - ❑ It requires your data to be moved multiple times
   - ❑ It requires CLI to use more system resource
   - ❑ It requires more complex code than do bound columns when performing fetches
   - ☑ All of the above

4. Which of the following is the best performing cursor sensitivity setting?

   - ❑ Sensitive
   - ❑ Insensitive
   - ☑ Asensitive


Now you can implement SQL Coding Techniques in your sleep.

## Conclusion

SQL performance on DB2 UDB for iSeries can be dramatically improved through careful utilization of the programming techniques provided in this course. The characteristics of many of the various SQL interfaces are similar, or the same. So, becoming thoroughly familiar with those properties will assist you in developing applications on DB2 UDB for iSeries that will efficiently deliver top system performance. Thank you for your interest in SQL for DB2 UDB for iSeries, and for participating in this online course.

**Good luck as you begin implementing SQL Coding Techniques for your organization.**

## Links

These Web sites and reference supplement the information in this course.

- **To view the *DB2 UDB for iSeries: SQL Performance Basics* online course:**
  **Ibm.com**/servers/enable/site/education/ibo/view.html?oc#db2
- **DB2 UDB for iSeries Web site**
  **ibm.com**/servers/eserver/iseries/db2/
- **IBM Redbooks:** redbooks.ibm.com
  -*Striving for Optimal Journal Performance* (SG24-6286)
  -Stored Procedures, Triggers, and UDFs for DB2 UDB for iSeries (SG24-6503-01)
  -Preparing for and Tuning the V5R2 Query Engine (SG24-6598)
- **Other IBM publications:**
  Publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/QB3AVG01/CCONTENTS
  *-DB2 UDB for iSeries SQL Programming Guide* (SC41-5611-03)
- **FAQs:**
  **-Java - iSeries Native JDBC Driver - FAQS**
  **ibm.com**/servers/enable/site/java/jdbc/jdbcfaq.html
  **-ODBC connection pooling FAQs**
  http://support.microsoft.com/default.aspx?scid=kb;en-us;169470
  **-SQL FAQs**
  **ibm.com**/servers/eserver/iseries/db2/sqlperffaq.htm
  **-CLI FAQs**
  **ibm.com**/servers/eserver/iseries/db2/clifaq.htm
  **-JDBC FAQs**
  **ibm.com**/servers/enable/site/java/jdbc/jdbcfaq.html
- **Query options (QAQQINI) file information:**
  **ibm.com**/servers/enable/site/bi/tuner/index.html
- **iSeries Information Center**
  **ibm.com**/iseries/infocenter
- **DB2 UDB for iSeries cursor sensitivity enhancements**
  **ibm.com**/developerworks/db2/library/techarticle/dm-0403milligan/index.html
- **DB2 Performance Workshop**
  **Ibm.com**/servers/eserver/iseries/service/igs/db2performance.html
- **Online courses, downloadable labs, and white papers:**
  **ibm.com**/servers/enable/site/education/ibo/view.html?oc#db2
- **Books:**
  **-SQL/400 Developer's Guide by Paul Conte & Mike Cravitz**
  as400network.com/str/books/Uniquebook2.cfm?NextBook=183
  **-SQL for eServer i5 and iSeries by Kevin Forsythe**
  www.mc-store.com/5063.html

## Trademarks



References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both: IBM, iSeries, eServer, OS/400, i5/OS, DB2, and Lotus.

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

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
Other company, product or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Some information addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Photographs shown are of engineering prototypes. Changes may be incorporated in production models.

## Test Questions

2. The DB2 UDB for iSeries runtime engine attempts to block reads and writes internally whenever possible, except during the following situations:
   - ❑ When the cursor is update or delete capable
   - ❑ When COMMIT (*CS) is specified
   - ❑ When ALWBLK (*ALLREAD) is NOT specified
   - ☑ All of the above

2. DB2 UDB for iSeries will only perform blocking when the application has specified a blocked SQL request.
   - ❑ True
   - ☑ False

3. SQL blocking of fetches can be improved most efficiently when you mix single and multiple row FETCH requests on the same cursor.
   - ❑ True
   - ☑ False

4. The DB2 UDB for iSeries resource is greatly reduced when you specifically list the columns that are required by the application, rather than simply coding SELECT *.
   - ☑ True
   - ❑ False

5. Using the FOR FETCH ONLY and FOR UPDATE OF clauses, help the optimizer better understand your SQL request by letting DB2 UDB for iSeries accurately know which cursors are:
   - ☑ Read-only or updateable
   - ❑ Validated by the precompiler
   - ❑ Required by the application
   - ❑ Write-only

6. When considering the use of VARCHAR columns in DB2 UDB for iSeries, if your primary goal is improving performance rather than saving memory, you'll want:
   - ❑ To set the ALLOCATE setting for a VARCHAR column to 0
   - ☑ The ALLOCATE value to accommodate 90-95 percent of the variable length values
   - ❑ The ALLOCATE value set equal to the VARGRAPHIC for primary key columns
   - ❑ All of the above

7. DDS-created tables in DB2 UDB for iSeries are faster on reads, and slower on writes, than SQL-created tables.
   - ❑ True
   - ☑ False

8. Good coding practice for DB2 UDB for iSeries requires that you NOT use compatible data types between the local host variables and table columns.

- ☑ True
- ❑ False

9. You should avoid using ILE SQL precompilers to create program objects directly since this interface defaults to Activation Group *NEW, creating extra performance overhead.
   - ☑ True
   - ❑ False

10. The benefit of stored procedures in distributed computing environments:
    - ❑ Allows the same amount of database work to be performed with fewer trips to the database server
    - ❑ Bundles together DB2 UDB for iSeries requests into a single stored procedure
    - ❑ Provides performance improvements further enhanced by the option of providing result sets back to ODBC and JDBC clients
    - ☑ All of the above

11. The Extended Dynamic package, when used in conjunction with the _____, can reduce line trips to the server by up to one third.
    - ❑ Host Server code
    - ☑ Cache Package Locally option
    - ❑ Parameter markers
    - ❑ SQL CONNECT statement

12. Usage of parameter markers is recommended for improving the performance of ODBC, JDBC, and CLI applications.
    - ☑ True
    - ❑ False

13. The extended dynamic package can become unusable if the package attributes don't match the application.
    - ☑ True
    - ❑ False

14. Sensitive is the Cursor Sensitivity setting that causes the best DB2 UDB for iSeries performance.
    - ❑ True
    - ☑ False

15. Reallocating the CLI environment causes the deletion of all active ODPs.
    - ☑ True
    - ❑ False

16. Which of the following is true about unbound columns?
    - ❑ It requires your data to be moved multiple times
    - ❑ It requires CLI to use more system resource
    - ❑ It requires more complex code than do bound columns when performing fetches
    - ☑ All of the above

17. SQLFetch can be done with multiple rows at the same time.
    - ❑ True
    - ☑ False

18. The JDBC preparedstatement class is more resource efficient, and has better chance for reusable ODPs, than the statement class.
    - ☑ True
    - ❑ False

19. Storing character data in which of the following encodings can improve the performance of JDBC applications?
    - Binary
    - ☑ Unicode
    - ❑ XML