SQL Statement Performance and Optimization

Topic Name	**SQL Statement Performance & Optimization**
Prepared By	**Atul Patel**
Prepared On	**February 2020**

Objectives

SQL Statement Performance and Optimization

In this module you will look at the factors that affect SQL performance and discover why organizations need to continually monitor, analyze and measure it.

Specific examples of SQL statements that can affect performance and how coding can improve them are provided in detail.

After completing this module, you should be able to:

- Describe Why and How SQL Statements Need to Be Continually Monitored for Performance
- Identify SQL Statements That Can Affect Performance

SQL Performance>

Introduction

As you have seen, there are many SQL statements that can be used to obtain, insert, update and delete data from a Db2 database. In most scenarios there are several different coding methods that can be used to create the same end result.

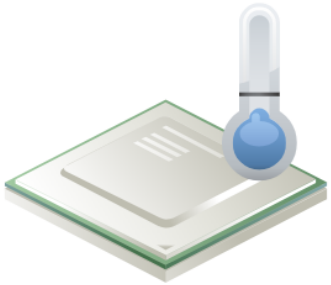
In many cases, as an application programmer you will not know whether your SQL code is performing at optimum performance, or may not even care as long as your code is working. In this section you will see why it is important that your code runs well and look at some of the tools that can be used for measuring, analyzing and reporting on SQL performance.

SQL Performance>Improving SQL Code

No matter how good you think your SQL code is, it can always be better. Environments are continually evolving with either new Db2 versions being introduced, or new systems and storage products being migrated to. Many of the new features introduced as a result of this will provide improved performance.

From a Db2 application programmer, tester or DBA perspective this means that SQL code needs to be constantly monitored and measured to determine that you are not wasting valuable Db2 processing resources.

Ensuring better SQL performance will result in the following:



Reduced CPU usage

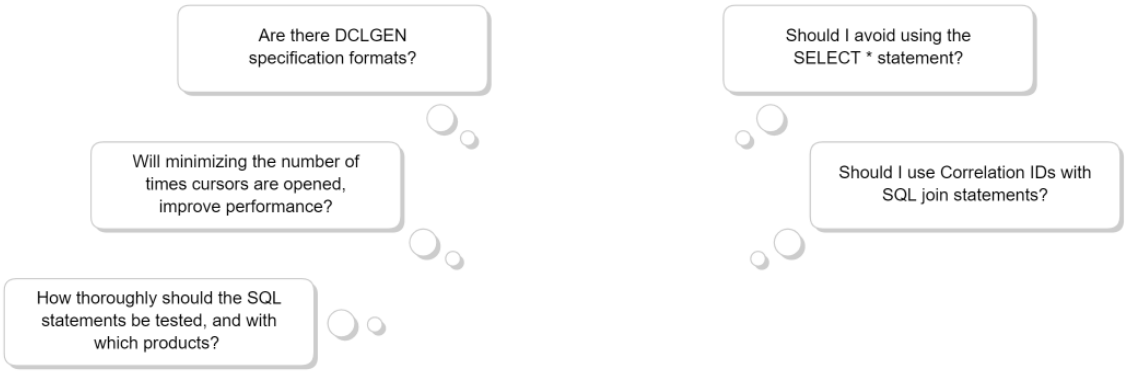
Better I/O utilization

Greater concurrency

SQL Performance>Standards

While later in this module you will be introduced to some SQL coding that may improve performance, it is important that you first identify whether your organization has any of their own SQL coding standards and try to understand why they have been implemented.

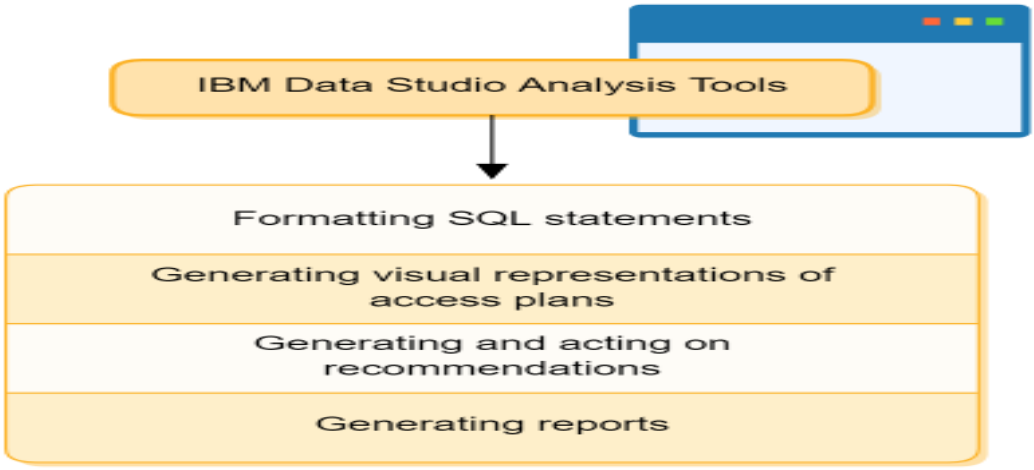
For example, your organization may have developed program templates that are provided to minimize coding errors and make maintenance easier. Testing of SQL statements will probably need to follow strict guidelines also.



SQL Performance>Tools for Measuring SQL Performance

Now that we have identified why measuring the performance of SQL processing is important, what tools can be used to capture this information and display it in a meaningful way?

Earlier in this course, several screens from IBM Data Studio were provided to display how it can access Db2 data. This product can also perform several other tasks associated with SQL performance and tuning, which are discussed on the following pages.

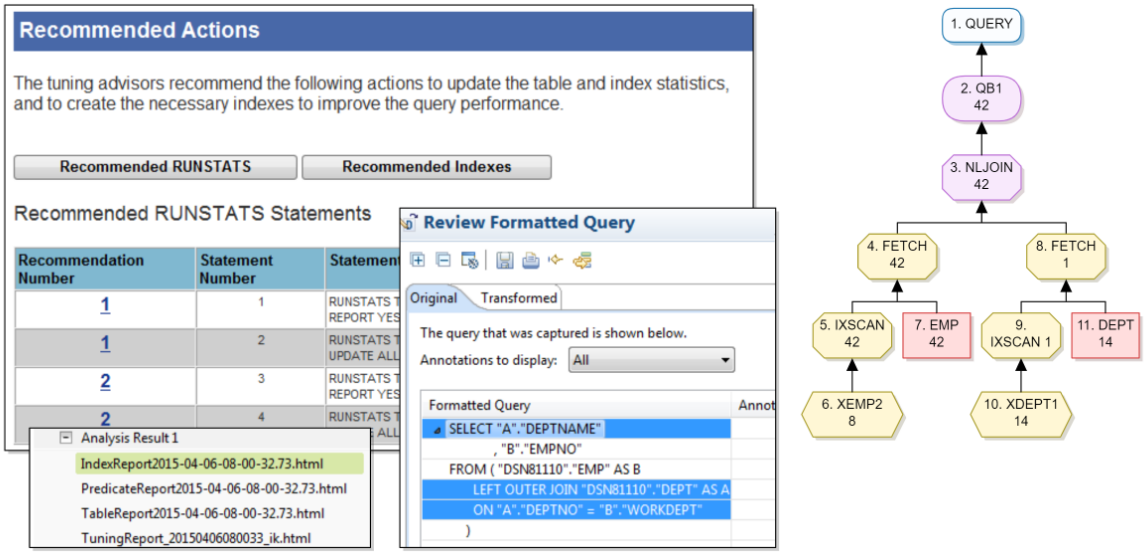


SQL Performance>IBM Data Studio

IBM Data Studio allows you to interrogate single SQL statements and report on a number of aspects including the following:

- How your statements can be formatted so that they are easier to read
- Graphical representation of access plans
- Statistical gathering recommendations
- Performance report generation

The following pages provide some introductory details describing how this information is obtained.

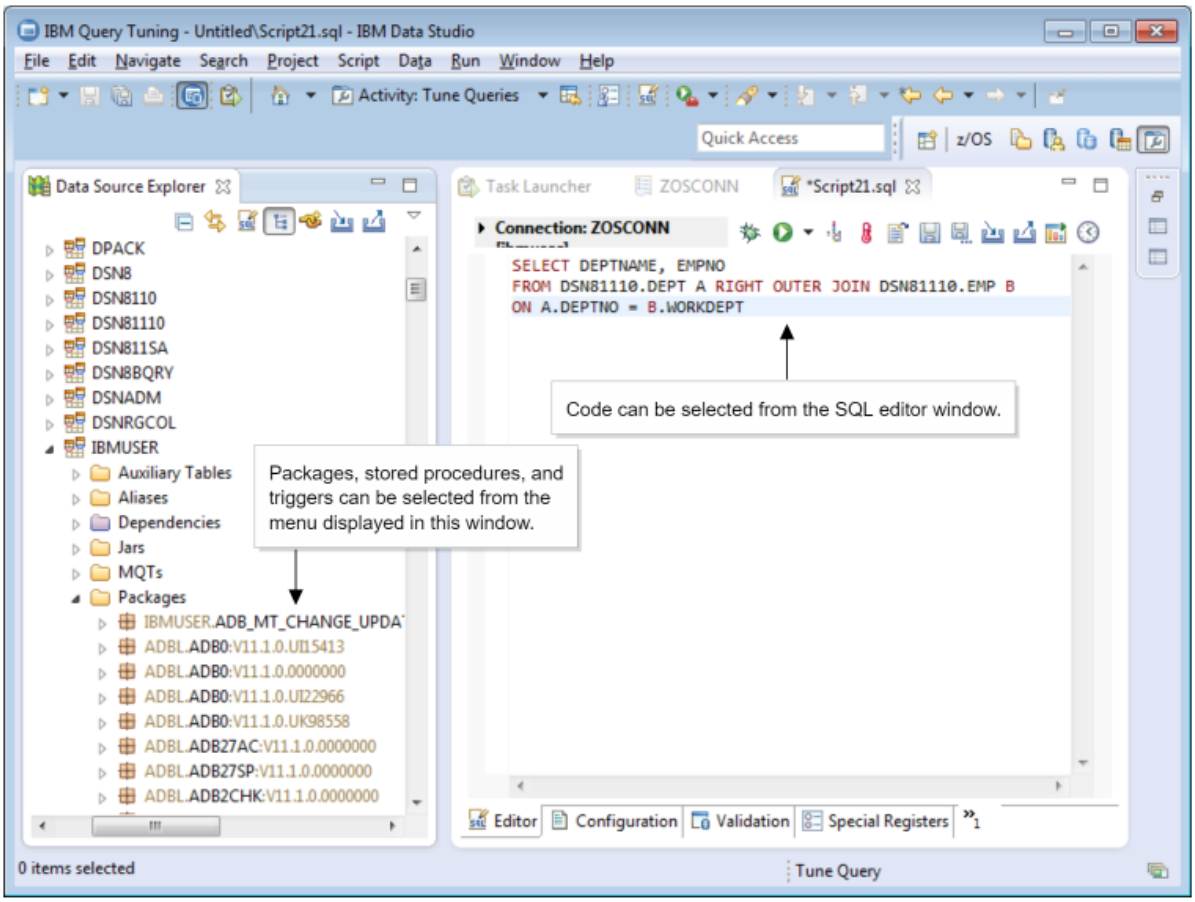


SQL Performance>IBM Data Studio Tuning Introduction

Before you can perform any of the tuning discussed on the previous page, you need to ensure the following:

- Your EXPLAIN tables have been created
- You have a connection to the required database
- The database has been configured for tuning, see your DBA for this
- You have the privileges that allow you to tune SQL statements

The first stage following this is to capture the SQL query that is to be tuned. In the Db2 for z/OS environment you are using here, you can reference the SQL from the SQL editor or Routine Editor, or from a package, SQL stored procedure, trigger, a user defined function with compiled SQL statements, or a view.



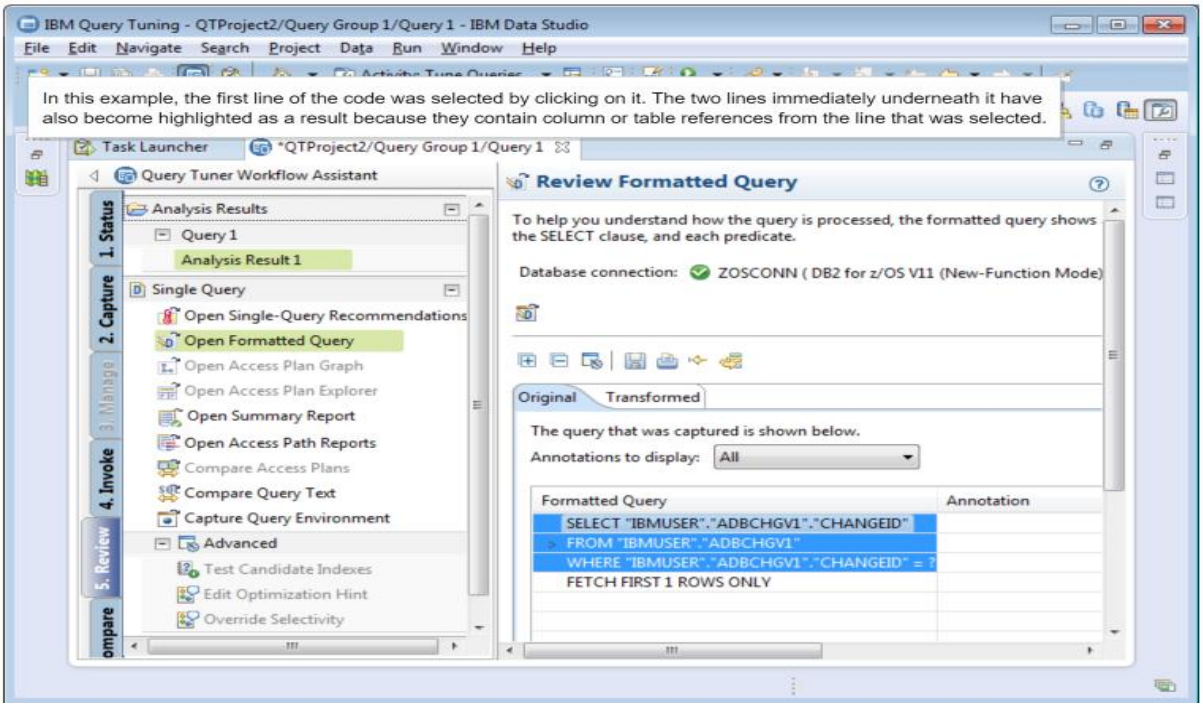
SQL Performance>Invoking the Query Tuner Workflow Assistant

In this example the Query Tuner Workload Assistant will be invoked by selecting a package from the Data Source Explorer window.

SQL Performance>Formatting the SQL Statement

There may be occasions where the person coding the SQL statements may not have done a particularly good in formatting it so it is easy for someone else to read. Selecting the Format SQL Statement option from the previous page will display the window shown here where each individual table reference, column reference and each predicate, is displayed on its own line.

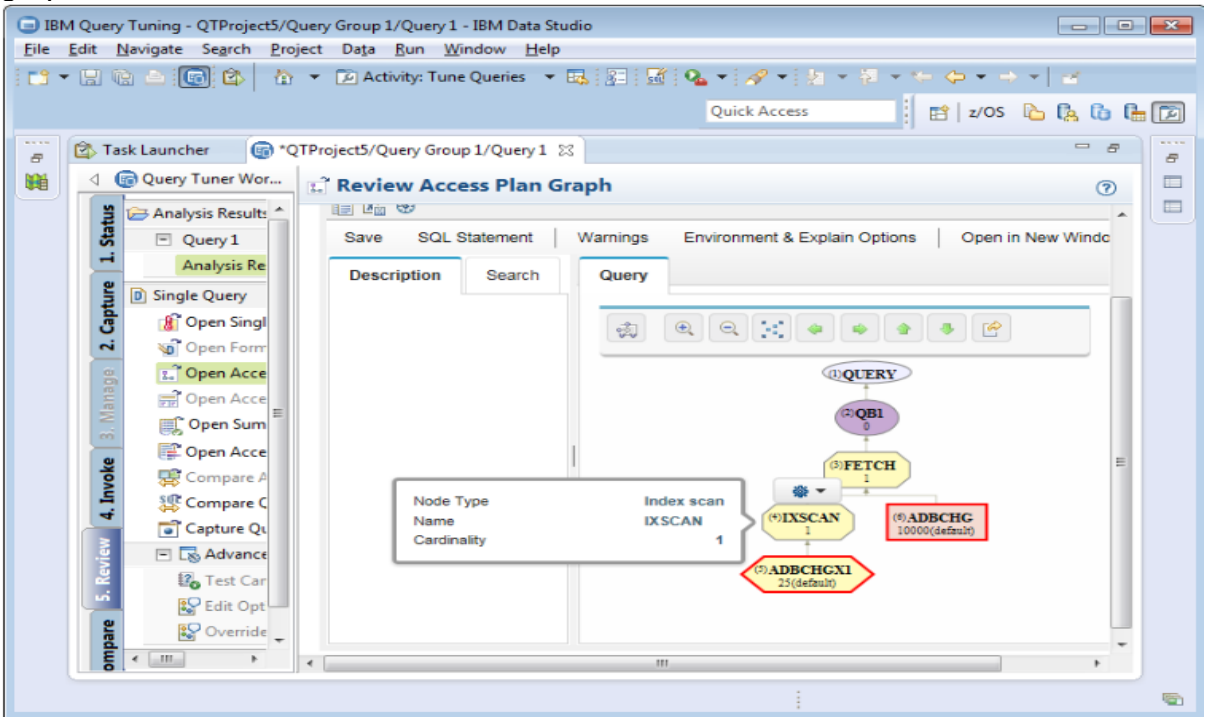
Complex queries contain parts that can be expanded and collapsed, and when clicking an individual line other related lines are also highlighted. The Transformed tab will display changes to the code as suggested by the Db2 optimizer.



SQL Performance>Display Access Plan Path

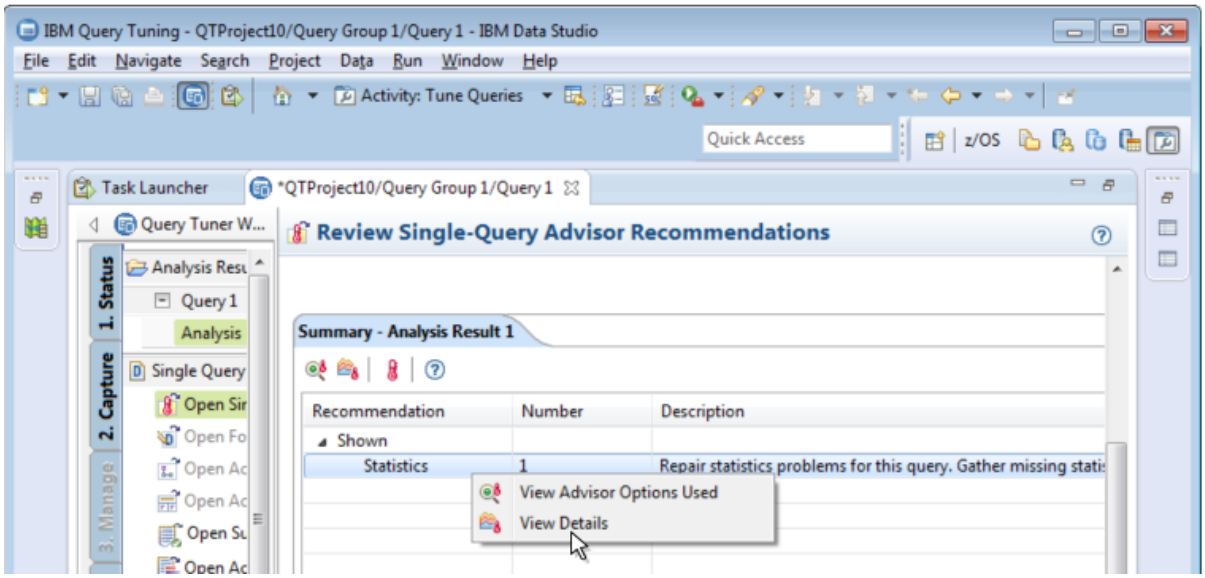
The screen displayed here will be shown if the Display Access Plan Path option was selected. It is a graphical representation of the query based on the Db2 optimizer's lowest cost of execution.

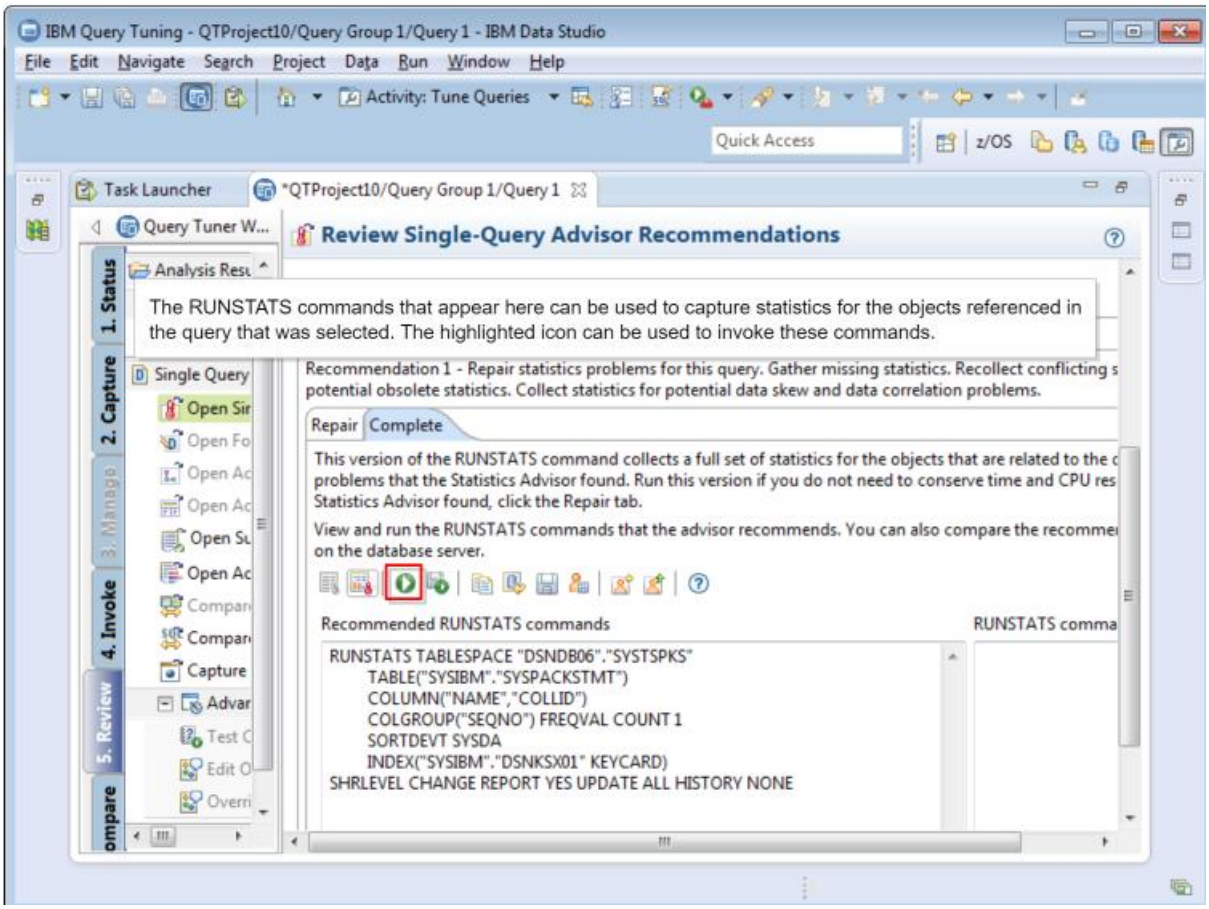
In the graphic, operators, SQL statements, and query blocks are all represented. Information about each of the nodes is available by rolling the mouse over them, as shown here. A description of the node's properties can be obtained by clicking the node. The resulting details will then be shown in the Description field to the left of the graphic.



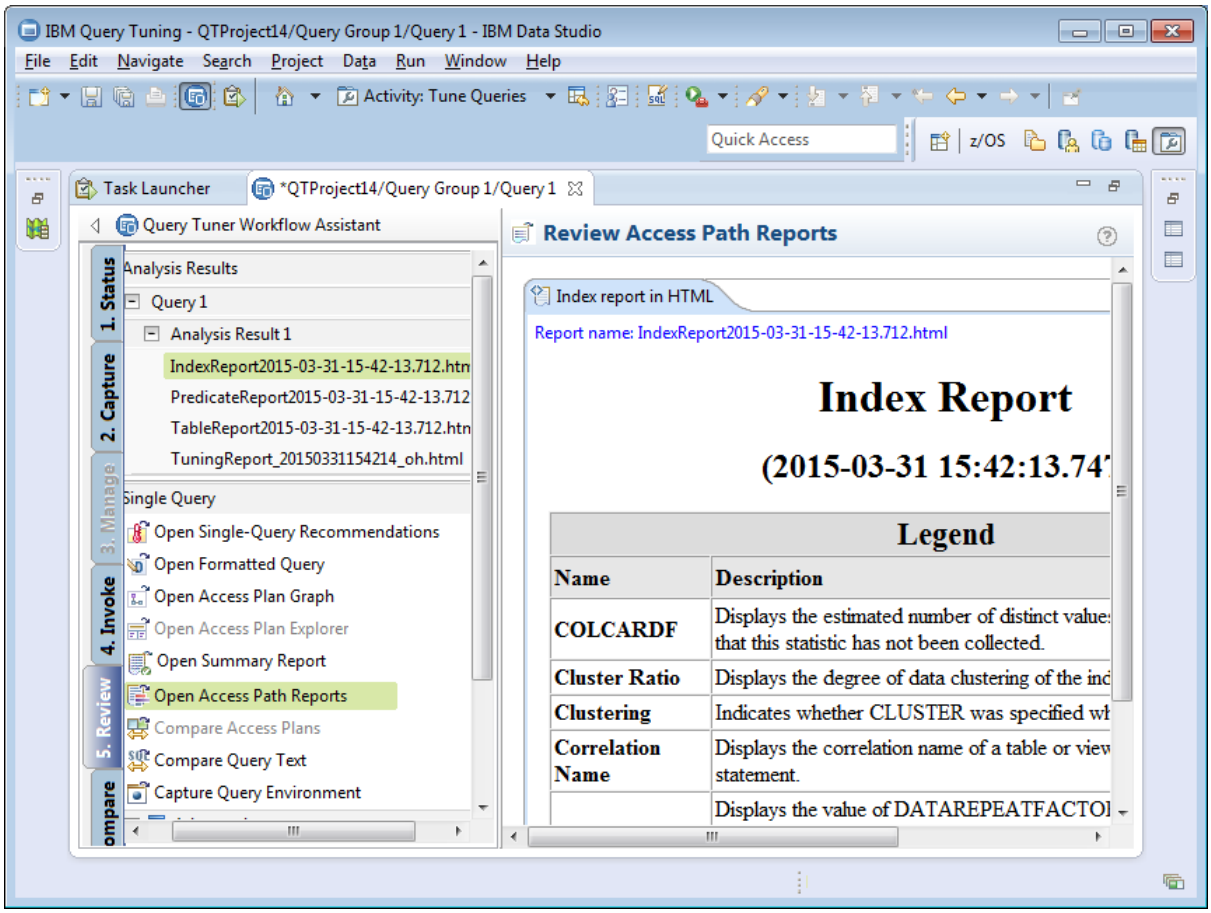
SQL Performance>Statistics

The screen displayed here will be shown if the Statistics option was selected. The Summary section displays recommendations on RUNSTATS commands that it considers need to be run in order to refresh or repair existing statistics. By updating these statistics the Db2 optimizer may choose a different access path.





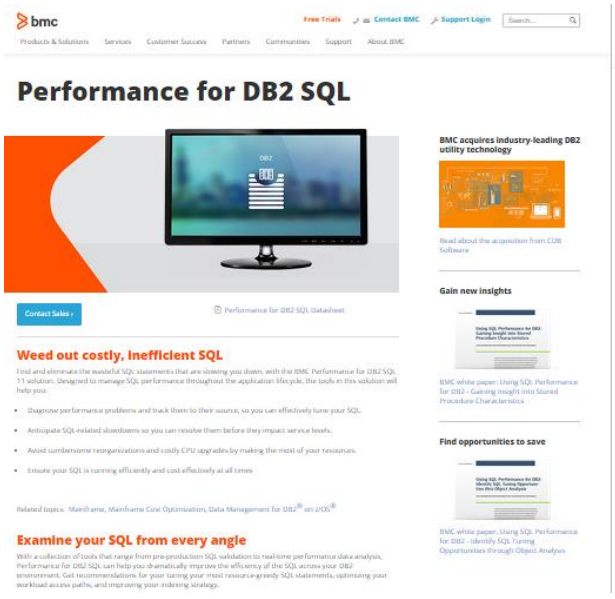
SQL Performance>Reports

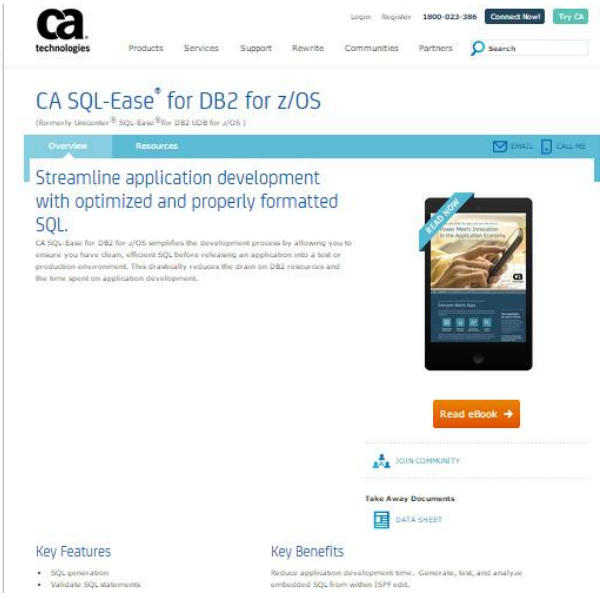


The screen displayed here will be shown if all the Generate Reports options were selected. You can see in the top left of the screen that four reports have been generated in html format.

The index report has been selected, with its details shown on the right of the screen.

SQL Performance>Other SQL Tuning Products





IBM Data Studio is not the only product that can be used to measure performance and provide tuning capabilities for SQL statements. There are many other products that can be used to perform similar tasks in a z/OS environment, including, CA SQL-Ease for Db2 for z/OS, and BMC Performance for Db2 SQL solution, which consists of Apptune for Db2, SQL Explorer for Db2, Performance Advisors for Db2, and Workbench for Db2.

Efficient SQL>

Introduction

Programmers should not only code to get the results they need, they also need to obtain that information as quickly as possible. This speed aspect is something that many programmers lose sight of, or are willing to put up with rather than have to spend time fine tuning the code. From an organizational perspective, running programs longer than what they need to is an inefficient use of resources, which evaluates to a cost value.

In this section you will look at a number of SQL coding examples and how they could be better designed to increase optimization.

Efficient SQL>SELECT Statement

In many examples shown in these Db2 courses, the SELECT * statement is used. In reality this should be avoided where possible and instead specify only the columns you require to obtain your end result.

Specifying SELECT * will result in all columns being selected, which may mean that a table scan is performed rather than an index scan if individual indexed columns were specified. As mentioned earlier, an index scan is quicker than a table scan.

```
SELECT * FROM DSN81110.SALESFACT
  FROM EMPLOYEE;
```

Query execution time => 873 ms

```
SELECT EMPNO, LASTNAME
  FROM EMPLOYEE;
```

Query execution time => 341 ms

Coding specific columns may result in the DB2 optimizer only needing to access the index, whereas requesting all columns may force DB2 to obtain the information from the data page, which will also increase I/O. Also be aware that any subsequent sorting that occurs will be on all columns, which can significantly increase processing times and costs.

Efficient SQL>Sorting

Use the GROUP BY and ORDER BY clauses only when sorting is absolutely necessary to your end result. Also remember that you don't need to code the sort column in the SELECT statement. This just increases the amount of processing unnecessarily.

```
SELECT EMPNO, LASTNAME
  FROM EMPLOYEE
   ORDER BY HIREDATE;
```

When this code is run it indicated a query execution time of 382 ms. When HIREDATE was added to the SELECT statement, the execution time was 404 ms.

Efficient SQL>Using DISTINCT

The SELECT DISTINCT statement is used to remove identical rows from the result table. In previous Db2 releases, this statement produced significant processing overheads but recent advances with the Db2 Optimizer have provided improvements in this area.

It is a popular statement with developers and it may be beneficial to measure the impact of DISTINCT statements against similarly written code, for your environment.

```
SELECT DISTINCT A.PRODUCTID, A.DESCRIPTION
  FROM DSN81110.PRODUCTS A, DSN81110.SALESFACT B
   WHERE A.PRODUCTID = B.PRODUCTID;
```

Query execution time => 692 ms

Below is another way that the query above could be rewritten

```
SELECT A.PRODUCTID, A.DESCRIPTION
 FROM DSN81110.PRODUCTS A
  WHERE EXISTS
     (SELECT 1 FROM DSN81110.SALESFACT B
      WHERE A.PRODUCTID = B.PRODUCTID);
```

Query execution time => 649 ms

Efficient SQL>Predicates

Predicates can be evaluated earlier in the processing depending on how they are coded and the structure of the data in the columns they are processing. Generally the earlier it can be processed the less amount of work that is required at a later stage. For this reason it is preferable if you can code predicates as stage 1 and indexable.

Predicates are processed as one of the following:

- Stage 1 indexable
- Stage 1 non-indexable
- Stage 2 non-indexable

Stage 1 indexable examples

```
COL = value
value BETWEEN COL1 AND COL2
COL BETWEEN value1 AND value2
COL LIKE 'pattern'
COL IS NOT DISTINCT FROM value
SUBSTR(COL,1,n) = value
DATE(COL) = value
YEAR(COL) = value
```

Stage 1 non-indexable examples

```
COL <> value
COL NOT BETWEEN value1 AND value2
COL NOT LIKE ' char'
COL IS DISTINCT FROM value
```

Stage 2 non-indexable examples

```
COL BETWEEN COL1 AND COL2
value NOT BETWEEN COL1 AND COL2
COL = (cor subq)
T1.COL1 <> T1.COL2
EXISTS (subq)
expression = value
```

Efficient SQL>Predicate Order

When writing your query you need to be aware that the sequence in which some of the predicates appear, can also affect its performance. The most restrictive predicates should appear first as this will filter out the most data in preparation for later statements that need to interact with that result set.

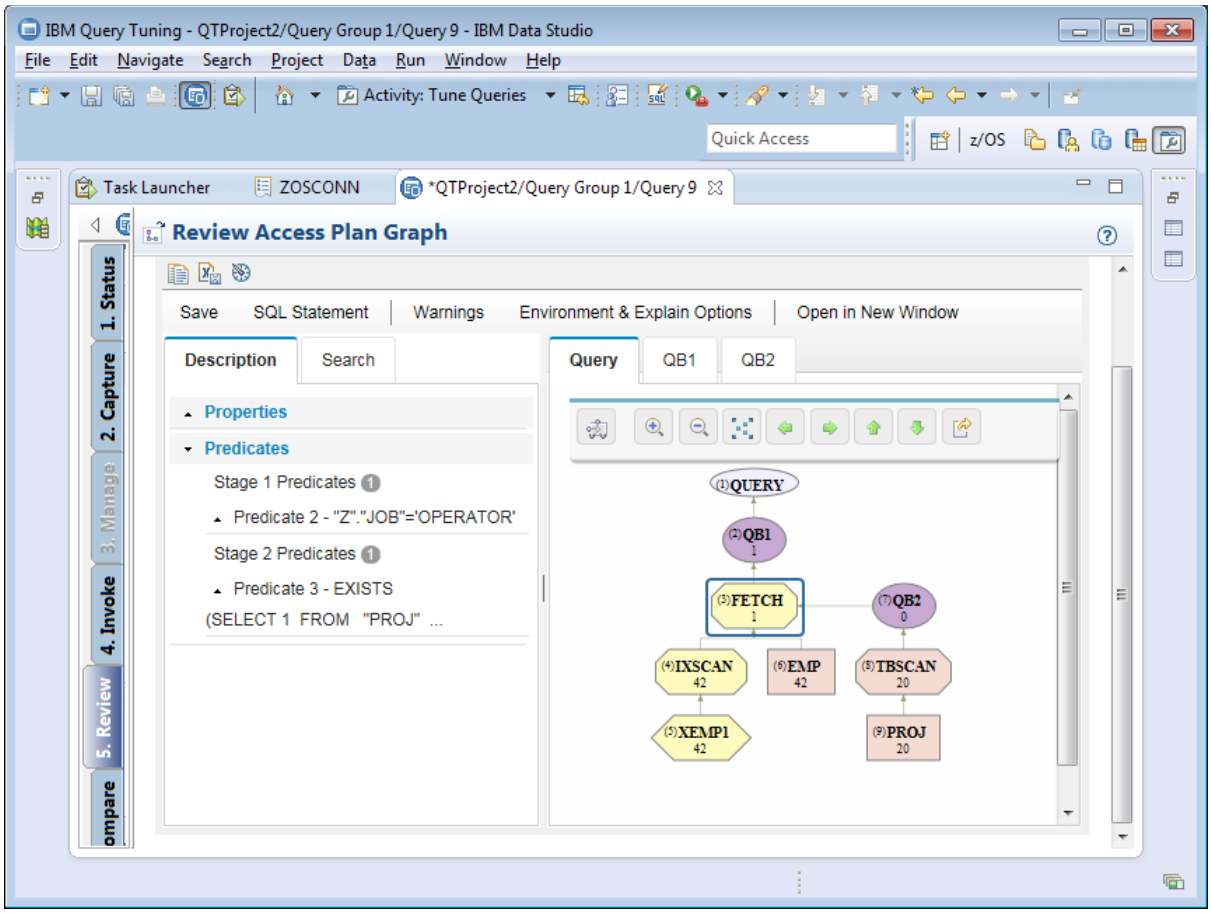
Note that the Db2 optimizer will select which predicates to process first anyway depending on the predicate's category, but if there are multiple predicates under the same category it will process them in sequential order.

Following stage 1 index predicate processing, any stage 1 non-indexable predicates will be applied in this sequence:

1. Equal predicates
2. Range predicates
3. In list predicates
4. Like predicates

Once these have been processed then stage 2 predicates will be applied.

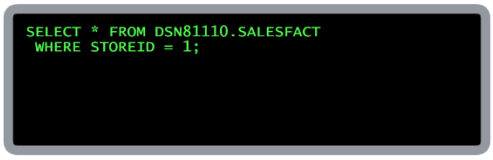
Efficient SQL>Identifying Predicate Stages



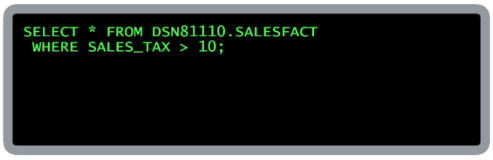
Using IBM Data Studio you are able to display predicate stage information when generating the access plan graph. In the example shown here, the FETCH node in the graphic has been selected and the details from this data retrieval operation are displayed on the left of the screen.

You can see that the FETCH process uses a stage 1 and stage 2 predicate. These items can be drilled down to display attributes associated with that predicate.

Efficient SQL>Indexable and Non-Indexable Predicates



In this example, the SALESFACT table has an index on the STOREID column, making this a more efficient query than if it did not have one.



In this query, the SALES_TAX column is not part of the index for the SALESFACT table, so this is a non-indexable predicate.

Because using indexes speeds up data access and processing, your SQL can be made more efficient by implementing indexable predicates and creating the related indexes.

Efficient SQL>Boolean Term Predicates

For SQL join operations, try to replace non-boolean term predicates with boolean ones. A Boolean term predicate is one that is connected by AND logic. When it is evaluated false for a particular row, makes the entire WHERE clause false for that particular row.

During a join operation a boolean predicate can reject rows at an earlier stage while non-boolean predicates can increase the amount of processing because rows cannot be immediately discounted.

```
SELECT PROJNO, EMSTDATE, EMENDATE FROM DSN81110.EMPPROJACT
WHERE (EMSTDATE > '2015-07-01')
OR (EMSTDATE = '2015-07-01' AND EMENDATE < '2015-10-01')
```

The query above contains non-boolean term predicates. It could be made more efficient and take advantage of the index on EMSTDATE by coding the boolean equivalent below.

```
SELECT PROJNO, EMSTDATE, EMENDATE FROM DSN81110.EMPPROJACT
WHERE (EMSTDATE >= '2015-07-01')
AND (EMSTDATE > '2015-07-01' OR
      (EMSTDATE = '2015-07-01'AND EMENDATE < '2015-10-01'))
```

Efficient SQL>Subquery Efficiency

You have seen on previous pages how many SQL statements can be written in another way in order to improve their performance. Subqueries, which are SELECT statements within a WHERE or HAVING clause, fall into this same category.

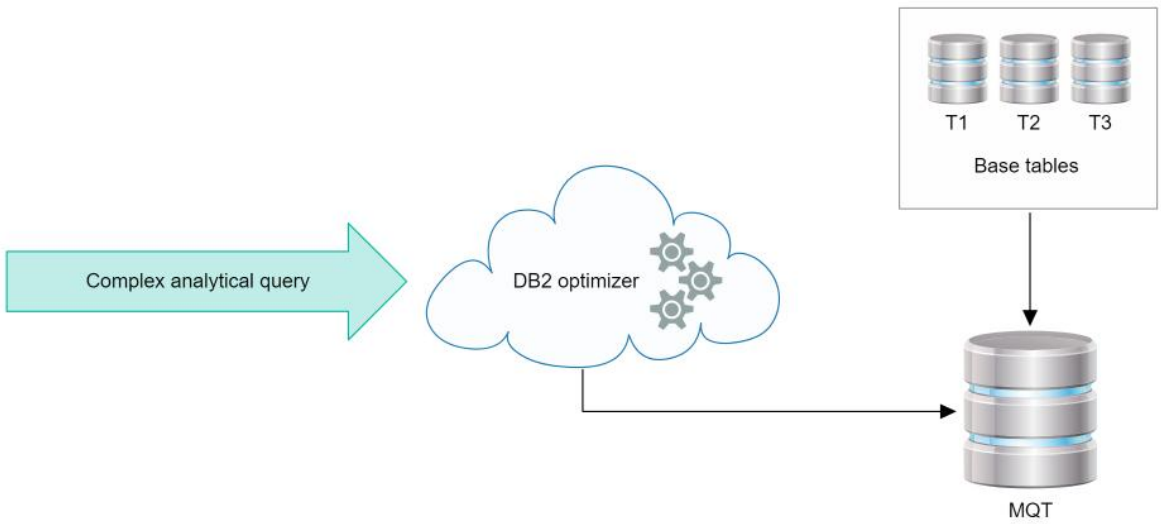
For example, the use of IN and EXISTS in subqueries can be used to produce the same results. The decision on which one to use is not as straightforward as it depends on the amount and type of data being accessed, and the access method chosen by the Db2 Optimizer. You should run the EXPLAIN facility against each type of subquery to determine its efficiency.

```
SELECT Z.PRODUCTID, Z.DESCRIPTION
 FROM DSN81110.PRODUCTS Z WHERE Z.PRODUCTID IN
    (SELECT Y.PID
     FROM DSN81110.PRODUCTSUPPLIER Y
     WHERE Y.SID LIKE 'A%')
```

These two subqueries will produce the same result but will be processed differently by DB2. Deciding which is most efficient will require some analysis of the data in the EXPLAIN tables.

```
SELECT Z.PRODUCTID, Z.DESCRIPTION
 FROM DSN81110.PRODUCTS Z
   WHERE EXISTS
   (SELECT 1 FROM DSN81110.PRODUCTSUPPLIER Y
   WHERE Y.PID = Z.PRODUCTID
   AND Y.SID LIKE 'A%')
```

Efficient SQL>Materialized Query Tables



For complex queries working with large amounts of data, you may want to consider using materialized query tables (MQT). These tables are basically a subset table created from one or more source tables and created in advance.

Using these instead of the source table can improve query performance and leave the base tables fully normalized.

Efficient SQL>Using Scrollable Cursors

Scrollable cursors can be very efficient if the result table is small and you are required to move backward and forward to access information. However, this type of processing is more labor intensive than using non-scrollable cursors, so you should test and analyze each situation to determine the most efficient process.

When using scrollable cursors:

- A scrollable cursor should be declared as SENSITIVE only if you need to view the most recent data
- Consider if the FETCH FIRST num ROWS ONLY can be used with the scrollable cursor
- When binding packages and plans that have updatable scrollable cursors, ensure that you specify ISOLATION(CS) and CURRENTDATA(NO)
- Consider making commits often

```
EXEC SQL
  DECLARE CURHAM SENSITIVE STATIC SCROLL CURSOR FOR
     SELECT EMPNO, LASTNAME, WORKDEPT
     FROM DSN81110.EMP
     WHERE EDLEVEL = 18
ENDEXEC
```

Efficient SQL>ROWSET Cursors

```
EXEC SQL
  FETCH NEXT ROWSET FROM PROJEX
    FOR :SIZE-ROWSET ROWS
    INTO :HV-PROJNO, :HV-ACTNO, :HV-EMSTDATE, :HV-EMPNO
END-EXEC.
```

The amount of rows requested can be specified or as in this example, the rowset size has been defined as a variable.

Fetching less than 5 rows	Performance could be worse
Between 10 and 100 rows	Best performance
Greater than 100 rows	Possibly no improvement

Using ROWSET positioning to fetch multiple rows is obviously efficient as only one operation is used to obtain many rows of data. How many rows to fetch at a time is a more difficult question to answer though as it will depend on the type of data and any action to be performed on it.

A good starting point is to obtain 100 rows, measure its efficiency and then add or remove a number from it and measure again.

Efficient SQL>Join Efficiency

```
SELECT DEPTNAME, EMPNO
FROM DSN81110.DEPT A RIGHT OUTER JOIN DSN81110.EMP B
ON A.DEPTNO = B.WORKDEPT
```

Note that when coding a right outer join that the DB2 Optimizer will automatically rewrite it as a left outer join. The code below shows how the code is transformed.

```
SELECT "A"."DEPTNAME"
     , "B"."EMPNO"
FROM ( "DSN81110"."EMP" AS B
       LEFT OUTER JOIN "DSN81110"."DEPT" AS A
       ON "A"."DEPTNO" = "B"."WORKDEPT"
     )
```

Performance can be affected by the order of tables when a join operation needs to be performed. While the Db2 Optimizer will choose the most efficient method for inner joins, regardless of how it is coded, for left and right outer joins you need to ensure that the main, or driver, table is referenced first.

If you find that the query is performing poorly then you may want to experiment with rearranging the order that tables are referenced.

Efficient SQL>Query Transformations

One last note about your SQL code.

As you have just seen, when looking at the EXPLAIN data your SQL may have changed slightly from what was originally coded. This is because Db2 can automatically identify from your code where it can improve access path efficiency and will rewrite the query.

Note that IBM Data Studio can display query transformations performed by the Db2 Optimizer.

The following are some of the actions that may occur as a result of a query transformation:

- Predicates that are pre-evaluated or are deemed not required, are removed
- Some new automatically generated predicates may be created
- Table references for some types of joins may be removed
- Subqueries may be converted to joins
- When using UNION ALL, subselects may be simplified or removed

Quiz

Question 1 of 2

Improving SQL query performance can result in which three items?

Select the correct options.

Click **Check My Answer** when you have finished.

- [ ] Larger database file creation
- [x] Reduced CPU usage
- [ ] Better auditing capabilities
- [x] Greater concurrency
- [ ] Faster recovery in the event of a disaster
- [x] Better I/O utilization

Question 2 of 2

Select **True** or **False** for each statement as it relates to SQL query performance monitoring and reporting, using IBM Data Studio.

Click **Check My Answer** when you have finished.

Tuning can be performed on SQL statements displayed in the SQL Editor window.

- (•) True
- ( ) False

SQL queries can be captured from packages, stored procedures and triggers.

- (•) True
- ( ) False

Transformation of SQL code performed by the Db2 Optimizer can be displayed.

- (•) True
- ( ) False

When performing tuning, selecting the Statistics option will display a graphical representation of the selected query.

- ○ True
- ◉ False

Question 1 of 3

Select True or **False** for each statement as it relates to improving SQL statement performance.

Click Check My Answer when you have finished.

Stage 2 predicates are processed before stage 1 predicates.

- ○ True
- ◉ False

IBM Data Studio can be used to identify whether a predicate is stage 1 or stage 2.

- ◉ True
- ○ False

When using the ORDER BY clause, the column specified must also appear in the SELECT statement.

- ○ True
- ◉ False

You should always attempt to code SELECT * even if you do not require data from all table columns.

- ○ True
- ◉ False

In a join operation, a boolean term predicate is more efficient than a non-boolean one.

- ◉ True
- ○ False

Question 2 of 3

Following stage 1 index predicate processing, any stage 1 non-indexable predicates will be applied in a particular order.

Equal predicates

1.

Range predicates

2.

In list predicates

3.

Like predicates

4.

Question 3 of 3

You need to run complex queries on large amounts of data. What type of table, which is a subset of one or more tables created in advance, can be produced in this scenario to improve query performance?

- ( ) XML tables
- ( ) Sample tables
- ( ) Explain tables
- (•) Materialized query tables

Module Test

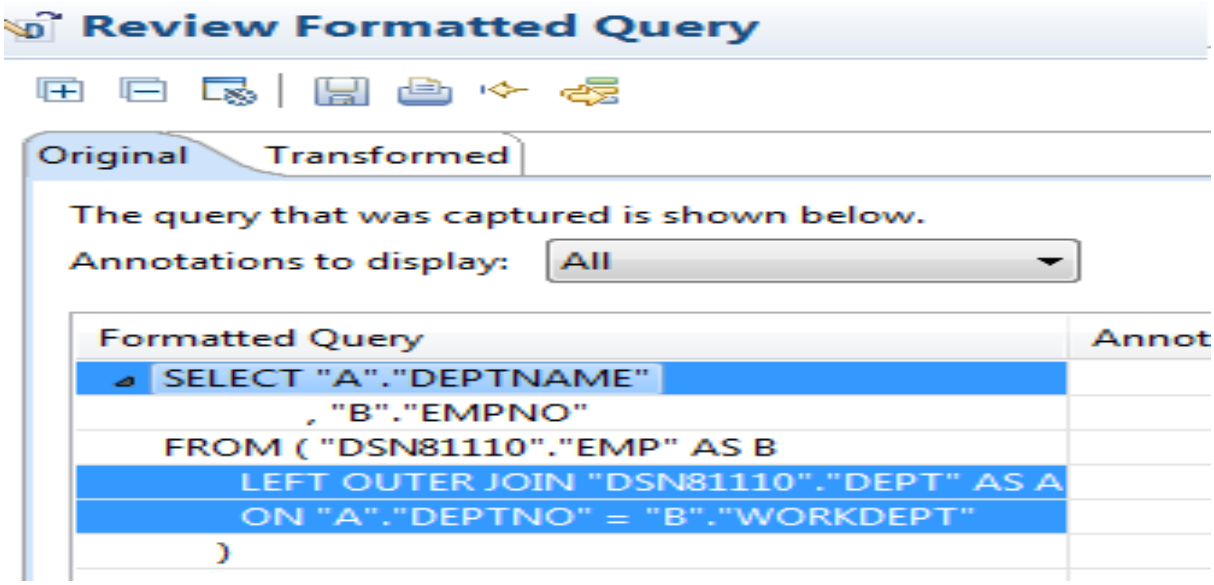
Question 1 of 7 - SQL Performance
Before undertaking any SQL query coding improvements on your own SQL, what do you need to check for?

- (•) Any organizational SQL standards
- ( ) That there are sufficient Db2 system backups
- ( ) Sufficient size buffer pools
- ( ) That you have access to the Db2 catalog

Question 2 of 7 - SQL Performance
You need to perform SQL query tuning using IBM Data Studio. Which four items need to be set up before this can be achieved?

- [x] EXPLAIN tables need to have been created
- [x] You need to have a connection to the required database
- [ ] Db2 Data Sharing needs to have been enabled
- [x] The database needs to have been configured for tuning
- [ ] Storage groups need to have been configured for tuning purposes only
- [x] You need to have privileges that allow you to tune SQL statements

Question 3 of 7 - SQL Performance
In the IBM Data Studio tuning screen displayed here, you have clicked on the first line of code, and others have become highlighted. Why has this occurred?



- (•) The clicked line specifies columns or tables that are referenced in the other highlighted lines
- ( ) It indicates that the clicked line and other highlighted lines contain errors
- ( ) Any lines not highlighted indicate that they are redundant and can be removed from the query

The clicked and highlighted lines indicate where the statements need to be rewritten

Question 4 of 7 - SQL Performance
The IBM Data Studio's Display Access Plan Path option provides you with a graphical representation of what?

- ( ) The environment in which the query will run
- (•) A query based on the Db2 Optimizer's lowest cost of execution
- ( ) The transformation changes suggested by the Db2 Optimizer
- ( ) The recommended RUNSTATS commands that you should use to collect or repair statistics

Question 5 of 7 - Efficient SQL
Why should the SELECT * code be avoided?

- ( ) Because it will lock the table for exclusive use for the entire processing time of the query
- ( ) Because it will not log access to the table in the audit log
- (•) Because every column will be selected from the specified table and this typically isn't what is required
- ( ) Because it will result in code being rebound every time it is run

Question 6 of 7 - Efficient SQL
The Db2 Optimizer will process predicates in a certain order. Which type of predicate will it attempt to process first?

- Stage 1 non-indexable
- Stage 2 non-indexable
- Stage 1 indexable
- Stage 2 indexable

Question 7 of 7 - Efficient SQL

You are testing your application program and using ROWSET positioning to fetch multiple rows. How many rows should you attempt to fetch initially in the FOR n ROWS statement?

- 100
- 1000
- 10000
- The maximum possible