

Coding Db2 Performance

By the Book!

Craig S. Mullins
Mullins Consulting, Inc.
15 Coventry Court
Sugar Land, TX 77479
www.mullinsconsulting.com





Mullins Consulting, Inc

Craig S. Mullins

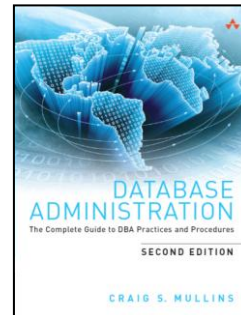
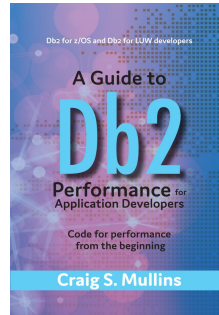
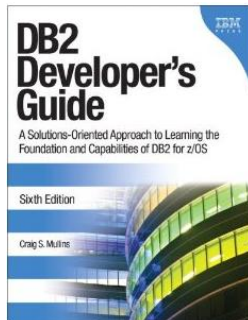
President & Principal Consultant

www.mullinsconsulting.com



Craig was named one of the
[Top 200 Thought Leaders in
BigData & Analytics](#) by
AnalyticsWeek.

Craig's current Db2 and DBA books



Details at: www.mullinsconsulting.com/books.html

This document is protected under the copyright laws of the United States and other countries as an unpublished work. This document contains information that is proprietary and confidential to Mullins Consulting, Inc., which shall not be disclosed outside or duplicated, used, or disclosed in whole or in part for any purpose other than as approved by Mullins Consulting, Inc. Any use or disclosure in whole or in part of this information without the express written permission of Mullins Consulting, Inc. is prohibited.

© 2019 Craig S. Mullins and Mullins Consulting, Inc. (Unpublished). All rights reserved.

Agenda



- ◎ Db2 performance from the application programmer point of view
- ◎ Based on my latest book: [A Guide to Db2 Performance for Application Developers](https://tinyurl.com/db2-craig)
- ◎ High-level walk-thru of the first 21 chapters of the book
- ◎ Quick overview of the remaining 8 chapters

<https://tinyurl.com/db2-craig>

The First 21 Chapters



- ◎ Chapter 1 – Defining Performance
- ◎ Chapter 2 – Code Relationally
- ◎ Chapter 3 – Minimize Data Access
- ◎ Chapter 4 – Avoid Black Boxes
- ◎ Chapter 5 – Concurrency
- ◎ Chapter 6 – Locking and Isolation
- ◎ Chapter 7 – Don't Code
- ◎ Chapter 8 – Indexing
- ◎ Chapter 9 – Clustering
- ◎ Chapter 10 – Optimization
- ◎ Chapter 11 – Seq vs. Random
- ◎ Chapter 11 – Seq vs. Random
- ◎ Chapter 12 – Joins
- ◎ Chapter 13 – Stages, Indexability and Sargability
- ◎ Chapter 14 – Filter Factors
- ◎ Chapter 15 – Access Paths
- ◎ Chapter 16 – Sorting & Grouping
- ◎ Chapter 17 – Parallelism
- ◎ Chapter 18 – Functions
- ◎ Chapter 19 – Stored Procedures
- ◎ Chapter 20 – Static vs. Dynamic SQL
- ◎ Chapter 21 – Many Ways to SQL

Chapter 1: What is Database Performance?

Database performance = the optimization of **resource** use to increase **throughput** and minimize **contention**, enabling the largest possible **workload** to be processed.

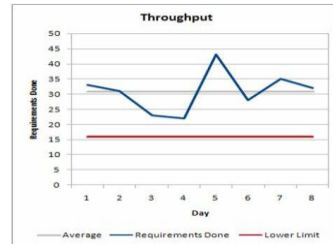
Workload

- Workload defines the demand. It is a combination of transactions, batch jobs, ad hoc queries, BI queries + analysis, utilities, and commands executing at any given time



Throughput

- Defines the overall capacity of the computer.
- It is a composite of I/O speed, CPU speed, parallel capabilities, and efficiency of system software.



Resources

- The hardware and software tools at the disposal of the system are known as the resources of the system




Contention

- Contention is the condition in which two or more components of the workload are attempting to use a single resource in a conflicting way.



Breaking Down Database Performance

- ◎ There are four major areas that must be in sync & tuned to deliver database performance:
 - ◎ The Application 
 - SQL and host language code
 - ◎ The Database
 - Database design, organization, indexing
 - ◎ The Subsystem or Instance
 - System parameters, memory pools, locking, distributed connections, etc.
 - ◎ The Environment
 - Operating system, network, storage, etc.

**But Let's
Simplify...**

**Db2
Performance
Really Boils
Down to Three
Things**

CPU



I/O



Concurrency



Chapter 2: Code Relationally

Learn what is meant by a relational database system.

The database is not a set of files.

- Files have no relationships set within and among them.



Tables are not files, they are based on sets.

- Sets are not ordered.
- Members of a set are all of the same type.
- When you perform an operation on a set, the action happens "all at once" to all the members of the set.

Rows are not records.

- Records are sequential.
- Rows have no physical order (conceptually).

Table

First Name	Last Name	Home #	Birthdate	Home #	Address
Sheeri	Cabral	121-5555	Sept 17, 1978	555-1212	123 Main Street, Boston MA
Tony	Cabral	121-6666	Jan 27, 1975	339-8682	500 College Ave, Boston MA
John	Smith	653-1210	April 7, 1998	485-2390	10 Wharf Road, Provincetown MA
Paul	Thompson	586-7987	May 27, 1996	626-5976	2 Harrison Street, San Francisco CA
Camille	Durand	871-5719	Mar 23, 1970		
Noelle	Durand	395-6161	July 6, 1960		
Raj	Sharma	168-5223	Aug 31, 1980		
Priya	Sharma	674-5391	Dec 4, 1979		

Columns are not fields.

- Columns are typed (and can be Null); not so for fields.
- Without a program a field has no meaning at all.

The Impact of Coding Relationally

- ◎ “Unlearn the “flat file” mentality!”
 - ◎ “Master file” processing is not appropriate for optimal DB2 applications!
 - This is where you open a file, read a record, then open another file and access a record in that new file using the data you just read.
 - Instead... Learn to Join!
 - ◎ Open Cursor is NOT the same as Open File
 - There is a lot going on “behind the scenes” when you open a cursor.
 - The cursor does not always read ALL of the data (usually) when you open the cursor
 - Depends on many things, but sorting is a big deal

The Impact on Modification When Coding Relationally

- ◎ The SQL Insert, Update, and Delete **operate on sets**, not individual rows or records.
 - ◎ Unless you have opened up a Cursor and issued the modification with WHERE CURRENT OF cursor
- ◎ Specifying appropriate Where clauses is VERY important to make sure that you are impacting only the data that is to be modified!

```
DELETE  
FROM EMP;
```

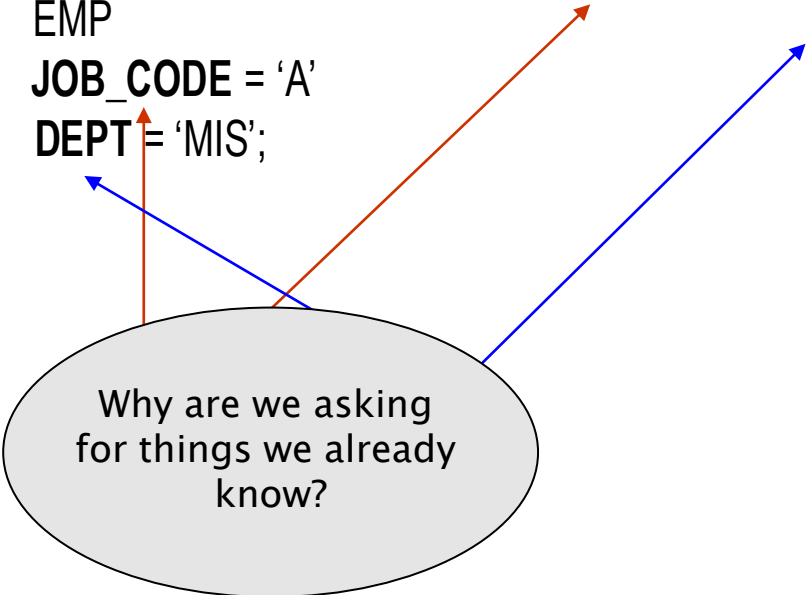
```
DELETE  
FROM EMP  
WHERE EMPNO = '00010';
```


Ask Only for What You Absolutely Need

- ⦿ Retrieve the minimum # of rows required
 - ⦿ Code appropriate WHERE clauses
 - ⦿ The only rows that should be returned to your program should be those that you need to process
- ⦿ Retrieve only the columns required - never more
 - ⦿ Don't ask for what you don't need
 - Sometimes shortened to → Avoid SELECT *
 - ⦿ This is a good idea (but it is not enough):
 1. Insulation of programs from change
 2. Performance

What is Wrong with this SQL?

```
SELECT LAST_NAME, FIRST_NAME, JOB_CODE, DEPT
FROM EMP
WHERE JOB_CODE = 'A'
AND DEPT = 'MIS';
```



Why are we asking
for things we already
know?

The diagram features a light gray oval callout box at the bottom center. From the top of this box, four arrows originate: two blue arrows and two red arrows. One blue arrow points to the 'JOB_CODE = 'A'' condition in the WHERE clause. The other blue arrow points to the 'DEPT = 'MIS'' condition. One red arrow points to the 'JOB_CODE' column in the SELECT list. The other red arrow points to the 'DEPT' column in the SELECT list. This visualizes the redundancy of filtering by JOB_CODE and DEPT when the columns are already being selected.

Matching Data Type and Length

- ◎ When coding predicates, it is a good practice to make sure the host variable data type/length matches the data type/length of the column
 - ◎ Or, for two columns, that the data type/length of each matches
- ◎ This used to result in immediate degradation
 - ◎ But no longer, so long as the arguments are within the same data type family
 - Character to character
 - Numeric to numeric
 - Date/time to date/time

Chapter 3: Minimize Passes Through the Data

- ⦿ Accessing the same data over and over consumes more resources
- ⦿ Organize your program to avoid issuing the same SQL over and over again... whenever possible
 - ⦿ So avoid processing **the same data** multiple times

```
SELECT firstnme, midinit, lastname, empno  
FROM emp  
WHERE workdept = 'A01' AND sex = 'F';
```

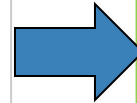
```
SELECT firstnme, midinit, lastname, empno  
FROM emp  
WHERE workdept = 'A01' AND YEAR(hiredate) > 1999;
```



```
SELECT firstnme, midinit, lastname, empno  
FROM emp  
WHERE workdept = 'A01'  
AND  
(sex = 'F' OR YEAR(hiredate) > 1999);
```


**Another
Possibility for
Reformulating
SQL to
Minimize
Passes Thru
the Data**

```
SELECT CREATOR, NAME, 'TABLE'
FROM SYSIBM.SYSTABLES
WHERE TYPE = 'T'
UNION
SELECT CREATOR, NAME, 'VIEW '
FROM SYSIBM.SYSTABLES
WHERE TYPE = 'V'
UNION
SELECT CREATOR, NAME, 'ALIAS'
FROM SYSIBM.SYSTABLES
WHERE TYPE = 'A'
ORDER BY NAME;
```



```
SELECT CREATOR, NAME,
CASE TYPE
WHEN 'T' THEN 'TABLE'
WHEN 'V' THEN 'VIEW '
WHEN 'A' THEN 'ALIAS'
END
FROM SYSIBM.SYSTABLES
ORDER BY NAME;
```

You Can Use CASE in UPDATE Statements, too!

Consider:

- ◎ Every employee with more than 18 years of education is to be given a 1% raise.
- ◎ But some departments have decided to give different raise amounts.

```
UPDATE emp
SET salary = CASE workdept
    WHEN 'A01'
        THEN salary+(salary*.05)
    WHEN 'C01'
        THEN salary+(salary*.025)
    WHEN 'D11'
        THEN salary+(salary*.10)
    WHEN 'D21'
        THEN salary+(salary*.75)
    ELSE
        salary+(salary*.01)
WHERE edlevel > 18;
```




Avoid Black Boxes

Reasons to Avoid Black Boxes

- ◎ Short cuts
- ◎ Extra code means extra work
- ◎ Ignorance of SQL is not a virtue
- ◎ SQL is already an access method
- ◎ Single point-of-failure



Chapter 5: Code for Concurrency

- ◎ Concurrency (along with I/O and CPU) is one of the three major factors that impact the performance of Db2 applications.
 - ◎ Recall that the goal is to reduce I/O and CPU usage, while increasing concurrency
- ◎ A lock management system is required to provide concurrent access
- ◎ Db2 supports locking at four levels, or granularities:
 - ◎ Table space, Table, Page, and Row
 - ◎ Db2 also provides LOB locking for large objects
 - BLOBs, CLOBs, and DBCLOBs

Timeouts



Lock timeouts can be a frequent cause of performance issues.

A timeout occurs when Db2 waits too long for lock (because another process holds an incompatible lock).

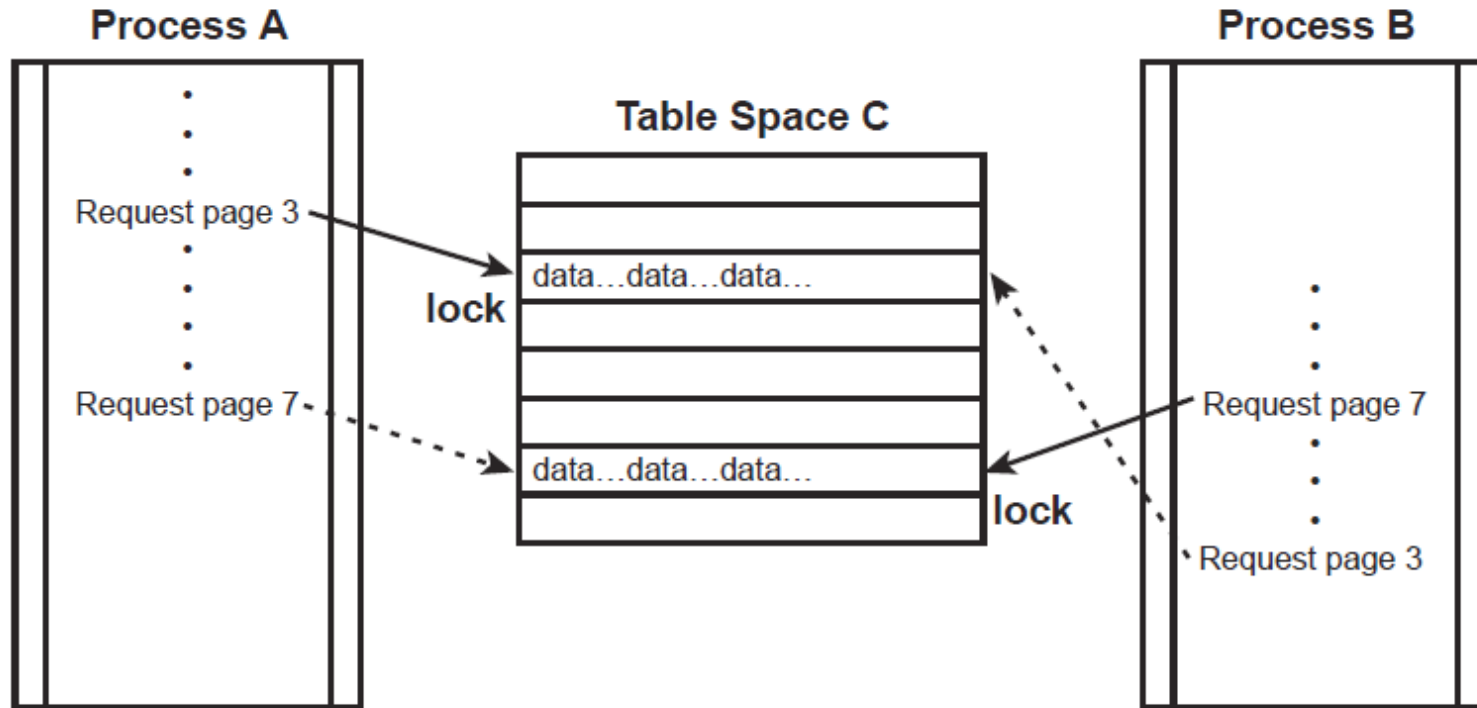
When your request times out it will either fail and must be rerun, or the request can be issued again if retry logic is coded into the program.

Timeouts increase wait time and reduce concurrency.

When you experience a timeout, another process holds a lock on the data that you are trying to modify.

So, you should try to *minimize the duration of locks* that are being held changes.

Deadlocks



Process A is waiting on Process B

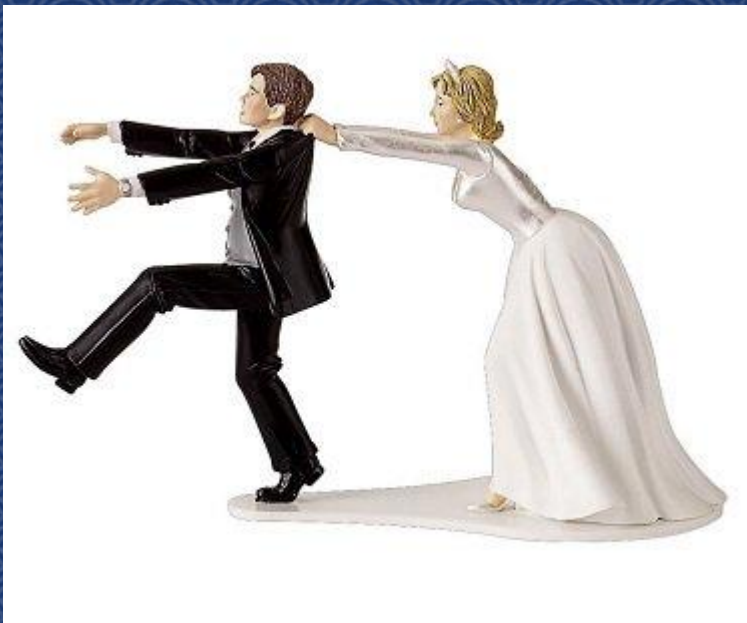
Process B is waiting on Process A

Tactics for Concurrency



- ⊙ Minimize deadlocks by coding updates in same sequence for all programs
- ⊙ Issue modification SQL statements as close to the end of the UOW as possible
 - ⊙ The later in the UOW the update occurs, the shorter the duration of the lock
- ⊙ Encourage Lock Avoidance
 - ⊙ ISOLATION(CS) / CURRENTDATA(NO) - can be used only with read only cursors
- ⊙ Use LOCK TABLE judiciously – *cautiously!*
- ⊙ Consider ISOLATION(UR) to avoid locking
 - ⊙ But be careful (more on Isolation levels coming up)
- ⊙ Exercise caution when using tools with Auto-Commit (e.g. TOAD, SPUIFI)

Avoid Bachelor Programming Syndrome



- Plan and implement a COMMIT strategy
Failure to do so will cause locking issues
- TIMEOUTs and DEADLOCKS

Fear of COMMITing

Chapter 6: Locking and Isolation Levels

- ◎ Allowing concurrent access requires a locking mechanism
 - ◎ Without locking, the following sequence would be possible:
 - PGM1 retrieves a row from the Employee table for Empno '000010'.
 - PGM1 issues an update statement to change that employee's salary to 55000.
 - PGM2 retrieves the Employee row for Empno '000010'. Because the change was not committed, the old value for the salary, 52750, is returned.
 - PGM1 commits the change, causing the salary to be 55000.
 - PGM2 changes a value (in a different column) and commits the change.
 - The value for salary is now back to 52750, negating the change made by PGM1.

Locking and Isolation Levels

Isolation levels can modify the way your program operates

- ◎ ***Isolation*** is a component of ACID. To achieve isolation a locking mechanism is required.
- ◎ At a high level it means that transactions can run at the same time. Any transactions running in parallel have the illusion that they are the only ones running (no concurrency).
- ◎ In other words, to the user, it appears that the system is running only a single transaction at a time.
- ◎ No other concurrent transaction has visibility to the uncommitted database modifications made by any other transactions.
- ◎ There are various levels of isolation that can impact data integrity if you do not understand how they work

The Isolation Levels

- You can specify the Isolation level for the entire program or at the SQL-statement-level.
 - CS: Cursor Stability
 - RR: Repeatable Read
 - RS: Read Stability
 - UR: Uncommitted Read

Cursor Stability (CS)

- ◎ CS is the Db2 implementation of the SQL standard read committed isolation level.
- ◎ CS is **probably the most common** Db2 isolation level in use in production applications because it offers a good **tradeoff between data integrity and concurrency**.
 - ◎ CS read-only **page locks** are released as soon as another page is accessed.
- ◎ When CS is specified the transaction will never read data that is not yet committed; **only committed data can be read**.
 - ◎ But if your program tries to **read the same data twice** there is no guarantee that it will be the same data.

Repeatable Read (RR)

- ◎ With RR isolation all page locks are held until they are released by a COMMIT (or ROLLBACK).
 - ◎ Potentially improved data integrity at the expense of less concurrency.
- ◎ An RR page locking strategy is useful **when an application program requires consistency in rows that may be accessed twice** in one execution of the program.
- ◎ **Example**: consider a reporting program that scans a table to produce a detail report, and then scans it again to produce a summarized managerial report. If the program is bound using CS, the results of the first report might not match the results of the second.

Read Stability (RS)

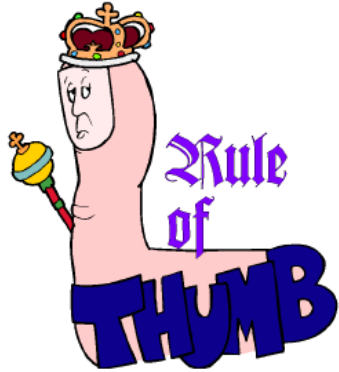
- ⦿ Read Stability (RS) is similar RR, but a little less.
- ⦿ A retrieved row or page is locked until the end of the unit of work; no other program can modify the data until the unit of work is complete, but other processes can Insert rows that might be read by your application if it accesses the row a second time.
- ⦿ Consider using RS over RR only when your program can handle retrieving a different set of rows each time a cursor or singleton SELECT is issued.
 - ⦿ If using read stability, be sure your application is not dependent on having the same number of rows returned each time.

Uncommitted Read (UR)

aka “Dirty Read”

- ◎ UR isolation level provides read-through locks, also known as *dirty reads*. UR can minimize concurrency problems, but at the cost of data integrity issues.
 - ◎ When you're using UR, an application program can read data that has been changed but is not yet committed.
- ◎ UR can be a performance booster because programs bound using UR will read data without taking locks.
 - ◎ Can cause data that is never in the database to be read and processed!
- ◎ **My experience:** UR is used too frequently.
 - ◎ With little concern for the possibility of impacting data integrity & accuracy

UR Usage



- ◎ The general rule of thumb is to **avoid** UR whenever the results must be **100 percent accurate**.
- ◎ Examples when UR is **not** a good idea:
 - ◎ Calculations that must balance are being performed on the selected data
 - ◎ Data is being retrieved from one source to insert to or update another
 - ◎ Production, mission-critical work is being performed that cannot contain - or cause - data integrity problems

Chapter 7: Avoid Writing Code (when you can)

Chapter 18: Functions

- ◎ Use built-in Db2 functionality instead of writing your own code!
 - ◎ Utilities
 - LOAD, UNLOAD, Export
 - ◎ Functions
 - BIFs and UDFs
 - ◎ Stored Procedures
 - ◎ Triggers
 - ◎ MERGE (UPSERT functionality)
 - ◎ SELECT FROM ...
 - INSERT | UPDATE | MERGE



There Are Many BIFs That Can Simplify Your Coding Efforts

Scalar Functions

Too many to list and discuss. Here are a few notable scalar functions:

- ◎ ABS
- ◎ CEILING
- ◎ COALESCE
- ◎ LOCATE
- ◎ LOWER, UPPER
- ◎ LPAD, RPAD
- ◎ LTRIM, RTRIM
- ◎ OVERLAY
- ◎ RAND
- ◎ TRUNCATE
- ◎ ...

Column Functions

- ◎ AVG
 - Returns integer
- ◎ COUNT
- ◎ COUNT_BIG
 - Returns
 - DECIMAL(31,0)
- ◎ COVARIANCE
- ◎ MAX
- ◎ MIN
- ◎ STDDEV
- ◎ SUM
- ◎ VARIANCE

Date/Time Functions

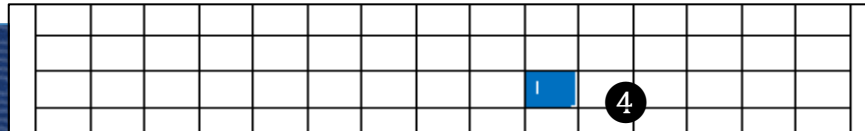
Too many to list and discuss. Here are a few notable ones:

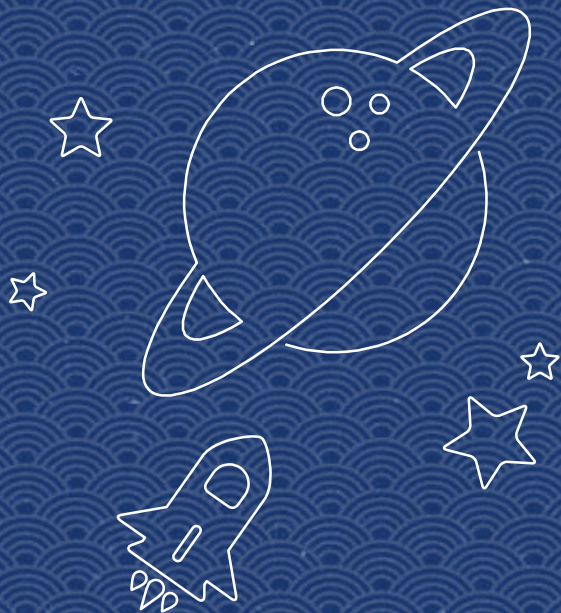
- ◎ DATE
- ◎ DAYS
- ◎ HOUR
- ◎ MINUTE
- ◎ MONTH
- ◎ MONTHS_BETWEEN
- ◎ NEXT_DAY
- ◎ QUARTER
- ◎ TIMESTAMP
- ◎ WEEK
- ◎ YEAR
- ◎ ...

Chapter 8: The Importance of Indexes

- ⦿ Perhaps the most important thing you can do to assure optimal Db2 application performance is to ensure that there are **appropriate indexes** to optimize your queries.
- ⦿ When you specify columns in the WHERE clause of your SQL queries, Db2 must search the data looking for rows that match.
- ⦿ Without indexes, all access to data would have to scan all available rows. Scans are very inefficient for very large tables.
- ⦿ Usually, Db2 can find the correct data much quicker using an index.

Find 63





What Indexes Exist?

Know the indexes available to you as you write your SQL so that your program will use indexes if possible

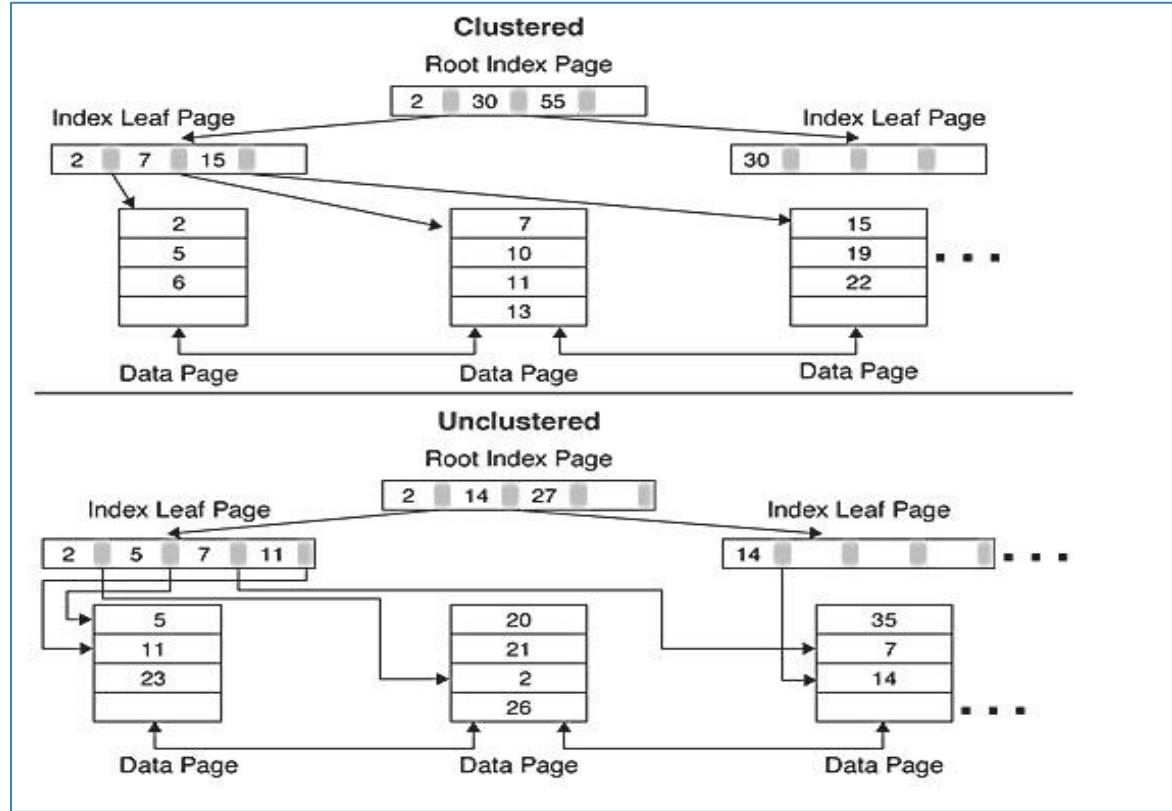
Information on indexes that exist can be found in the Db2 Catalog

Ask your DBA for help if you do not have access

Physical Storage

- ⊙ Clustering determines the way that data is stored physically, on disk.
 - ⊙ Clustering controls and sequences rows on disk, contiguously by key values.
 - ⊙ It will have an impact on the way that you write your SQL queries.
- ⊙ Clustering is controlled in Db2 by means of an index.
 - ⊙ The left-to-right order of columns (as defined in the index) defines the collating sequence for the data.
 - ⊙ There **can be only one clustering sequence** per table (because physically the data can be stored in only one sequence).

Clustered versus Unclustered Data





- ◎ Relational database systems differ significantly from traditional file-based systems
 - ◎ When accessing files the programmer decides what is accessed how and in what order.
 - ◎ When accessing data using SQL from an RDBMS (like Db2) the Relational Optimizer decides what is accessed how and in what order.

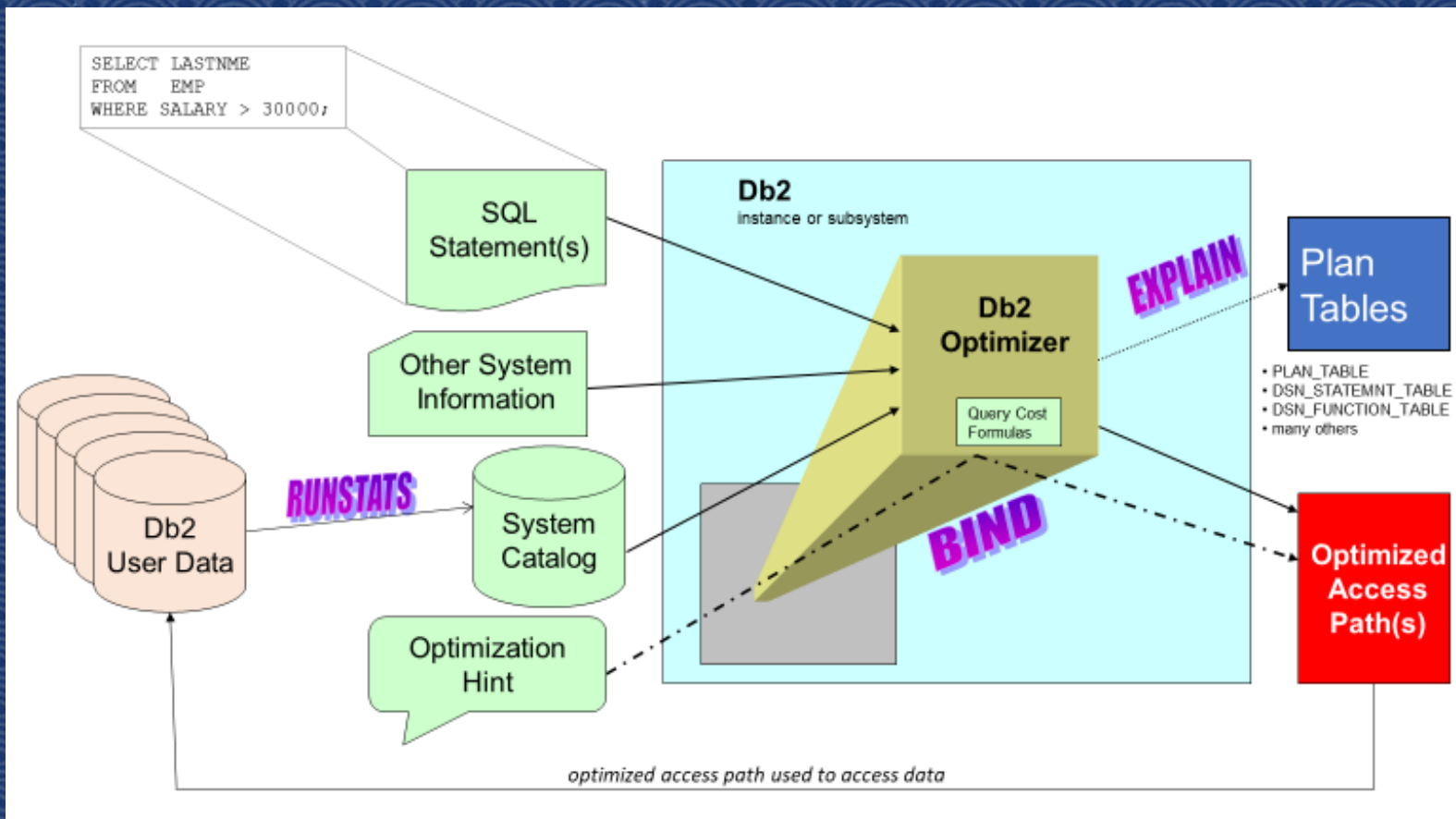
Physical Data Independence

- ◎ You use SQL to specify what to retrieve, not how to retrieve it.
 - ◎ Doesn't matter how data is physically stored. If indexes are removed, DB2 can still access the data (less efficiently). If a column is added to the table, the data can still be manipulated by Db2 without changing the program code.
 - ◎ This is possible because the physical access paths to Db2 data are not coded by programmers in application programs but are generated by Db2.
- ◎ Compare this with older, legacy data manipulation mechanisms (such as VSAM, IMS, and flat files).

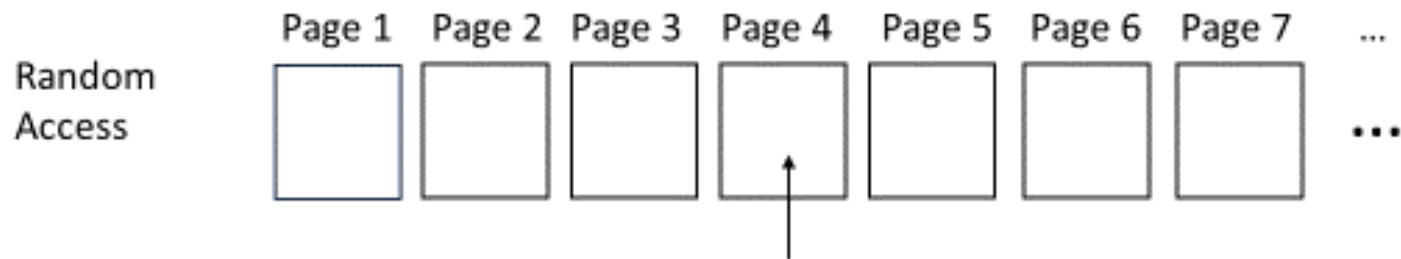
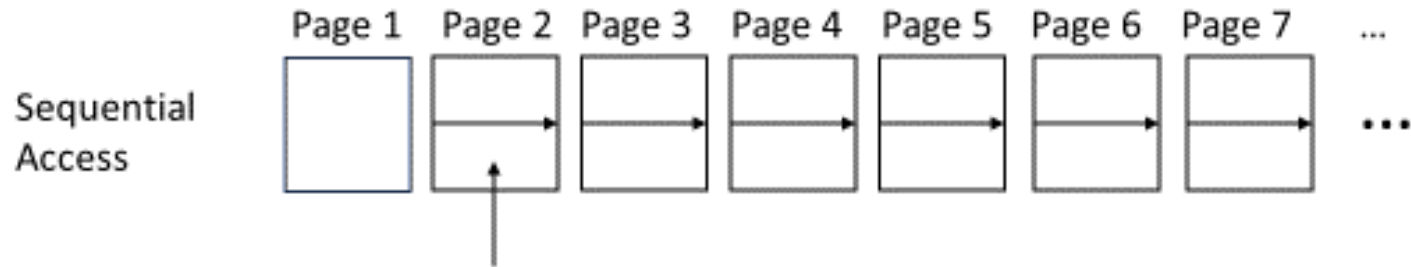
What is the Db2 Optimizer?

- ◎ The Db2 Optimizer basically works like an expert system.
 - ◎ An expert system is a set of standard rules that – when combined with relevant data – can return an expert opinion.
- ◎ The Db2 Optimizer renders expert opinions on data retrieval methods for SQL queries based on the relevant data stored in its system catalog.
 - ◎ Statistics – as populated with RUNSTATS

The Db2 Optimizer – A Visual Representation



Chapter 11: Sequential and Random Data Access



Sequential and Random Data Access (and clustering)

Random Access

- Get a row by means of a key value

```
SELECT FIRSTNME, LASTNAME, SALARY
FROM   EMP
WHERE  EMPNO = '00010';
```

**Which one
will benefit
most from
clustering?**

Sequential Access

- Get multiple rows by range

Multiple values apply with =

```
SELECT FIRSTNME, LASTNAME, SALARY
FROM   EMP
WHERE  DEPTNO = 'A01';
```

BETWEEN, >, >=, <, <=

```
SELECT FIRSTNME, LASTNAME
FROM   EMP
WHERE  SALARY > 25000.00;
```

Single Table Access Paths



Table and Table Space Scans

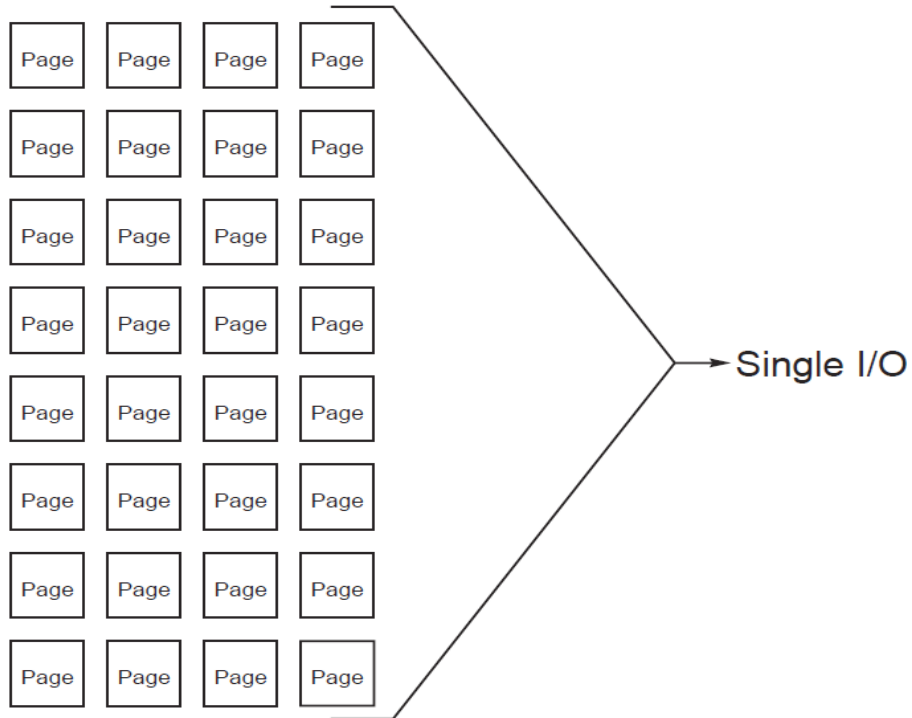
- ⦿ A scan reads every data page, examines each row on the page to determine whether it satisfies the query predicates and return only those rows that match.
 - ⦿ Sequential prefetch and sequential detection can access pages before they are needed, priming the pump and making a scan more efficient (next slide).
- ⦿ Avoid table space scans for large tables (usually)
 - ⦿ Requires accessing every page/row; or at least every page/row in a partition for a partition scan
 - ⦿ Indexed access usually outperforms scans: unless all (most) pages/rows need to be read or the table is small

Why Sequential Prefetch and Detection Can Help

Page

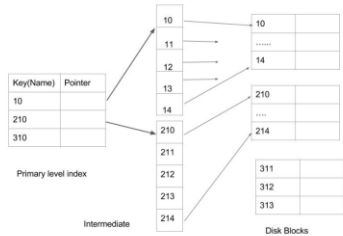
Single I/O

Normal I/O occurs one page at a time.



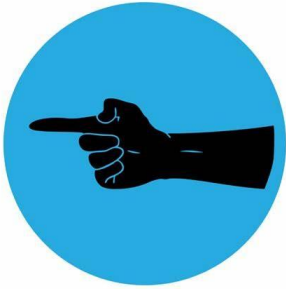
With sequential prefetch, up to 32 pages
can be retrieved with a single I/O.

Single Table Access Paths - Indexed



- ◎ *Indexed Access*
 - ◎ **Direct index lookup.** Fastest way to retrieve a single row
 - ◎ **Matching index scan**
 - ◎ **Non-matching index scan**
 - ◎ **Index screening**
 - ◎ **Multiple index access**
 - ◎ **One Fetch index access**

Direct Row Access



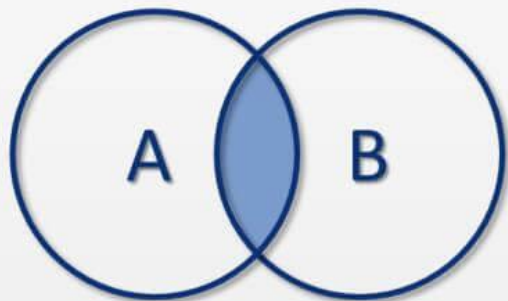
DIRECT

- ◎ **Direct row access** is another very efficient data access.
 - ◎ If an application selects a row from a table that contains a Rowid column, the row ID value implicitly contains the location of the row.
 - ◎ If you use that row ID value in the search condition of subsequent Select, Delete, or Update operations, Db2 might be able to use direct row access to navigate directly to the row.
- ◎ Although direct row access is very efficient, it is not very practical because you must provide the Rowid value in the query, which is usually not known to the user or programmer.

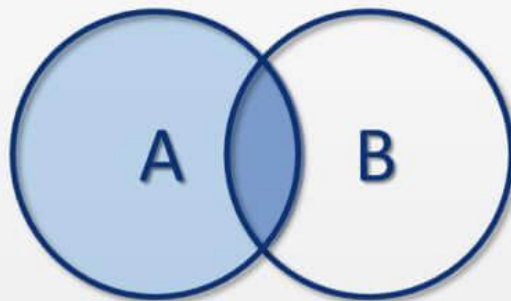


- ◎ In SQL, a join combines columns from one or more tables
 - A table can be joined to itself
 - The join criteria need not be equality ($<$, $>$, etc.)
- ◎ Inner Join – most common
 - Creates a result table by combining column values of the joined tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied by matching non-NULL values, column values for each matched pair of rows of A and B are combined into a result row.
- ◎ Outer Join (Left, Right, and Full)
 - The joined table retains each row—even if no other matching row exists.

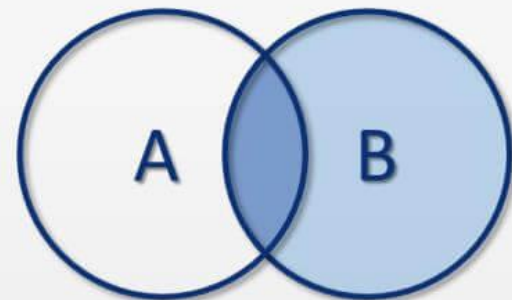
INNER JOIN



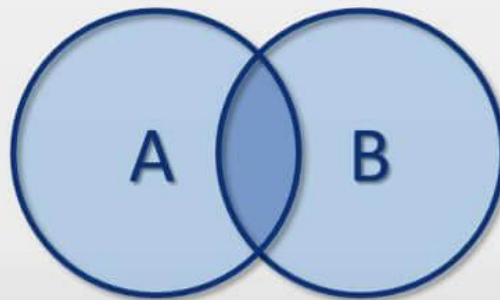
OUTER JOIN



LEFT



RIGHT



FULL

**Make Sure You
Include
Everything
Necessary**

- ⊙ Always provide join predicates (i.e. no Cartesian products)
- ⊙ If Joining TableA to TableB to TableC then make sure there are appropriate predicates
 - $A \rightarrow B \rightarrow C$ or $A \rightarrow C \rightarrow B$

```
SELECT E.Empno, E.Fname, E.Lname, E.Address, D.Deptname
FROM   Employee   E,
       Department D
WHERE  E.Workdept = D.Deptno
```

Not a Cartesian Product

Types of Joins

- ◎ Each multi-table query is broken down into separate access paths.
 - ◎ The optimizer selects 2 tables & creates an access path for that join.
 - ◎ Not done randomly, but based on what will be optimal.
 - ◎ Then it goes to the next table... and so on.
- ◎ There are three join methods (...well 4)
 - ◎ Merge-Scan Join
 - ◎ Nested Loop Join
 - ◎ Hybrid Join (Db2 z/OS)
 - ◎ Hash Join (Db2 LUW)

Basics of Joining

- ⊙ There are certain basics that are common to each join method.
 - ⊙ First decision is which table should be processed first. This table is referred to as the *outer table*. After this, a series of operations are performed on the outer table to prepare it for joining.
 - ⊙ Rows from the outer table are then combined to the second table, called the *inner table*. A series of operations are also performed on the inner table either before, during or after the join.
 - ⊙ Although all joins are composed of similar steps, each of the join methods is quite different when you get beyond the generalities.

Inner versus Outer?



- The smaller the table the more likely it will be the outer table. This reduces the number of times the inner table is re-accessed.
- If selective predicates can be applied, the table is more likely to be chosen as the outer table –
 - ...because the inner table is only accessed for rows satisfying the predicates applied to the outer table.
 - In most cases the table with fewest duplicates will be chosen as the outer table in a join.
- If index lookup can be done on one of the tables, it is a good candidate to use as the inner table.

Types of Join Methods

- ⦿ Nested Loop Join
- ⦿ Merge Scan Join
- ⦿ Hybrid Join (z/OS only)
- ⦿ Hash Join (LUW only)

Nested Loop Join

- ◎ To perform a *nested loop join (NLJ)*, a qualifying row is identified in the outer table, and then the inner table is scanned searching for a match.
 - ◎ A qualifying row is one in which the predicates for columns in the table match.
- ◎ When the inner table scan is complete, another qualifying row in the outer table is identified. The inner table is scanned for a match again, and so on. The repeated scanning of the inner table is usually accomplished with an index to minimize I/O cost.

Merge Scan Join

- ⊙ With the *merge scan join (MJ)*, the tables to be joined need to be ordered by the join predicates.
- ⊙ Each table must be accessed in order by the columns that specify the join criteria. This ordering can be the result of either a sort or indexed access. After ensuring that both the outer and inner tables are properly sequenced, each table is read sequentially, and the join columns are matched up. Neither table is read more than once during a merge scan join.

Hybrid Join (Db2 for z/OS only)

- ◎ For DB2 for z/OS there is the *hybrid join*, which uses data & pointers to access and combine the rows from the tables being joined.
 - ◎ You can think of it as a combination (or hybrid) of nested-loop and merge join techniques. The high-level steps employed by the hybrid join are:
 1. Outer table rows are either retrieved in sequence using an index or the qualifying values are retrieved and sorted.
 2. For each unique value in the outer table, a matching index scan is used to find RIDs for rows of the inner table.
 3. Partial rows of qualifying RIDs are built from the inner table along with the qualifying columns from the outer table using work table spaces.
 4. The partial rows are sorted in RID sequence to eliminate duplicate RIDs.
 5. Process the inner table with list prefetch.

Hash Join (*Db2 for LUW only*)

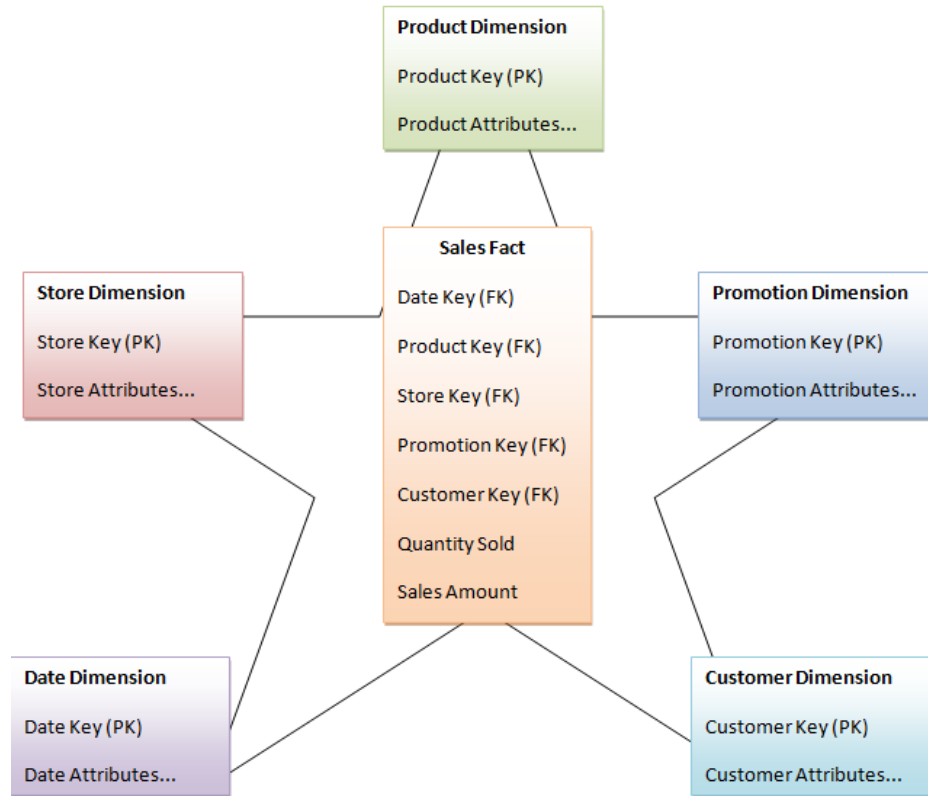
- ◎ For Db2 for Linux, Unix, and Windows, there is the *hash join*. Hash join requires one or more predicates of the form:

`table1.ColX = table2.ColY`

- ◎ ...and for which the column types are the same.
- ◎ The inner table is scanned & the rows copied into memory buffers from the sort heap allocation. The memory buffers are divided into partitions based on a “hash code” computed from the column(s) of the join predicate(s).
- ◎ If the size of the first table exceeds the available sort heap space, buffers from selected partitions are written to temporary tables. After processing the inner table, the outer table is scanned and its rows are matched to the inner table rows by comparing the “hash code.”

Also Star Join

*For data
warehousing
and BI type of
queries*



Which Join Method When?

- ⦿ In general, **NLJ** is preferable when a **small number of rows qualify** for the join.
- ⦿ As the number of qualifying **rows increases**, the **merge** join becomes a better choice.
- ⦿ In the case of a hash join, the inner table is kept in memory buffers. If there are too few memory buffers, then the hash join is obliged to spill. The optimizer attempts to avoid this and so will pick the smaller of the two tables as the inner table, and the larger one as the outer table.

Chapter 13: Stages, Indexability and Sargability



SARGable:
Has a Search
Argument to
query with

- ◎ Db2 for z/OS – Stage 1 and Stage 2;
Indexable and nonindexable
- ◎ Db2 for LUW – Data sargable and residual;
Range-delimiting and index sargable
- ◎ Indexable
 - Does not mean that an index WILL be used
 - Just that an index CAN be used
- ◎ This information is documented in the Db2 manuals:
 - Db2 12 for z/OS: Managing Performance (SC27-8857)
Page 359...
 - Db2 11.1 for LUW: Performance Tuning
Page 258... Table 57

Build Better Programs by Avoiding Stages 3 & 4

1...2...
3...

- ◎ We learned about Stages 1 (data sargable) and 2 (residual), with Stage 1 being more efficient than Stage 2. Now let's learn about Stages 3 and 4.
 - ◎ Stage 3 can be thought of as moving predicates from SQL into your programs.
 - ◎ Stage 4 can be thought of as the black box we talked about earlier.*
 - ◎ If the data has to be brought into the program before it is filtered out, performance will suffer!
 - Stage 1 better than Stage 2
 - Stage 2 better than Stage 3
 - Stage 3 better than Stage 4

Chapter 14: Filter Factors

- ◎ Filter factor is a ratio (a number between 0 and 1) that estimates I/O costs.
 - ◎ It is an estimate of the proportion of table rows for which a predicate is true.
 - ◎ Depending on the type of predicate, the optimizer plugs statistics from the Db2 Catalog into a filter factor formula to estimate the number of rows that can qualify for that particular predicate.

Lower filter factor is more efficient

Filter Factor Formulas - *a sampling*

Predicate Type	Formula	Default Filter Factor
COL = value or COL = :hostvar	$1 / \text{FIRSTKEYCARD F}$.04
COL <> value or COL <> :hostvar	$1 - (1 / \text{FIRSTKEYCARD F})$.96
COL IN (list of values)	$(\text{list size}) * (1 / \text{FIRSTKEYCARD F})$	$.04 * (\text{list size})$
COL NOT IN (list of values)	$1 - ((\text{list size}) * (1 / \text{FIRSTKEYCARD F}))$	$1 - (.04 * (\text{list size}))$
COL IS NULL	$1 / \text{FIRSTKEYCARD F}$.04
COL IS NOT NULL	$1 - (1 / \text{FIRSTKEYCARD F})$.96
COLA = COLB	$1/\text{FIRSTKEYCARD F}$ [COLA or COLB] <i>whichever is smaller</i>	.04
COL < value ... <= ↗	$(\text{LOW2KEY-value})/(\text{HIGH2KEY-LOW2KEY})$.33
COL > value ... >= ↖	$(\text{HIGH2KEY-value})/(\text{HIGH2KEY-LOW2KEY})$.33
COL LIKE '%char' or COL LIKE '_char'	1	1

Filter Factor Example

- Consider the following query:

```
SELECT Empno, Lastname, Sex
FROM   Emp
WHERE  Workdept = 'A00';
```

- What is the filter factor?
 - From the table on the previous slide:
1/FIRSTKEYCARD F
 - Assume FIRSTKEYCARD F is 9... making the FF 1/9
or .1111
 - Db2 assumes 11% of the rows from this table will
satisfy the request

Chapter 16: Access Paths and Explain

- ◎ The Explain command is used to show the access paths chosen by the DB2 optimizer for SQL stmts.
 - ◎ Explain populates a series of tables with access path information and those tables can be queried, enabling us to see the access paths that will be used.
 - ◎ The subject of an Explain can be a single SQL statement, or multiple SQL statements in a program, or even the dynamic statement cache.
 - ◎ The Explain tables are standard Db2 tables that must be defined with predetermined columns, data types, and lengths. The exact definitions are slightly different for Db2 for z/OS and Db2 for LUW.

EXPLAIN Tables – z/OS vs. LUW

For Db2 for z/OS, sample DDL for the Explain tables in the member DSNTESC of the SDSNSAMP library. DB2 12 has 20 explain tables and can also be created by executing a new stored procedure called
`ADMIN_EXPLAIN_MAINT.`

The tables are similar but can differ, so always check to ensure that you have the right version of the Explain tables for your Db2, system, and version.

For Db2 for LUW, you can use the `SYSPROC.SYSINTALL OBJECTS` procedure to create the Explain tables. If you would like to review the DDL for the Explain tables, it can be found in the directory:
`INSTHOME/sqlllib/misc`

Running Explain

- ⦿ Explain can be run in the same manner as any other SQL statement.
- ⦿ To Explain a single SQL statement, precede the SQL statement with the Explain command as follows:

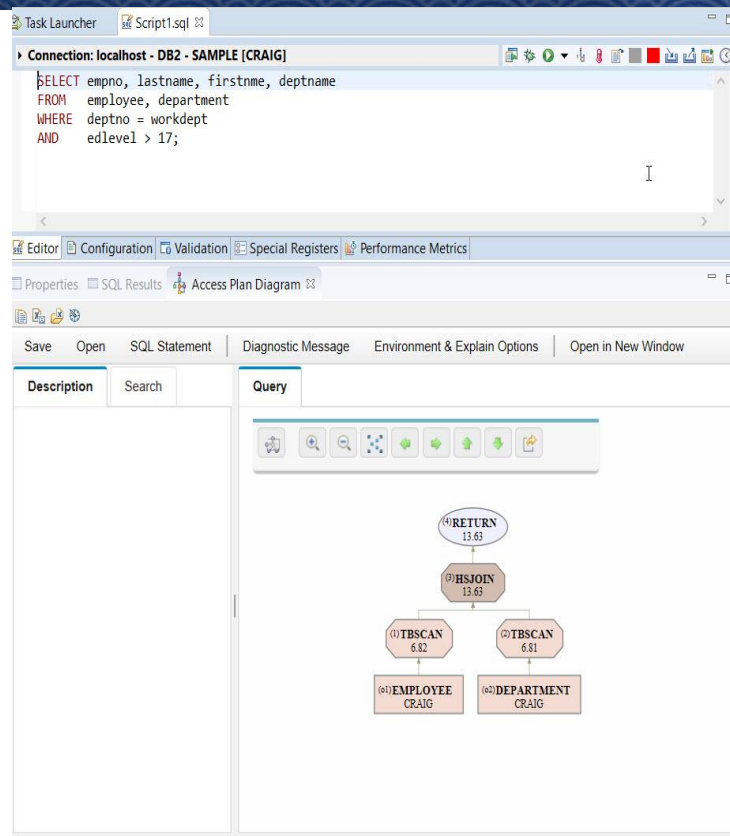
```
EXPLAIN ALL SET QUERYNO = n FOR  
      SQL statement;
```

- ⦿ You can also Explain an entire program by specifying EXPLAIN(YES) on when you Bind the program

Interpreting Explain Output

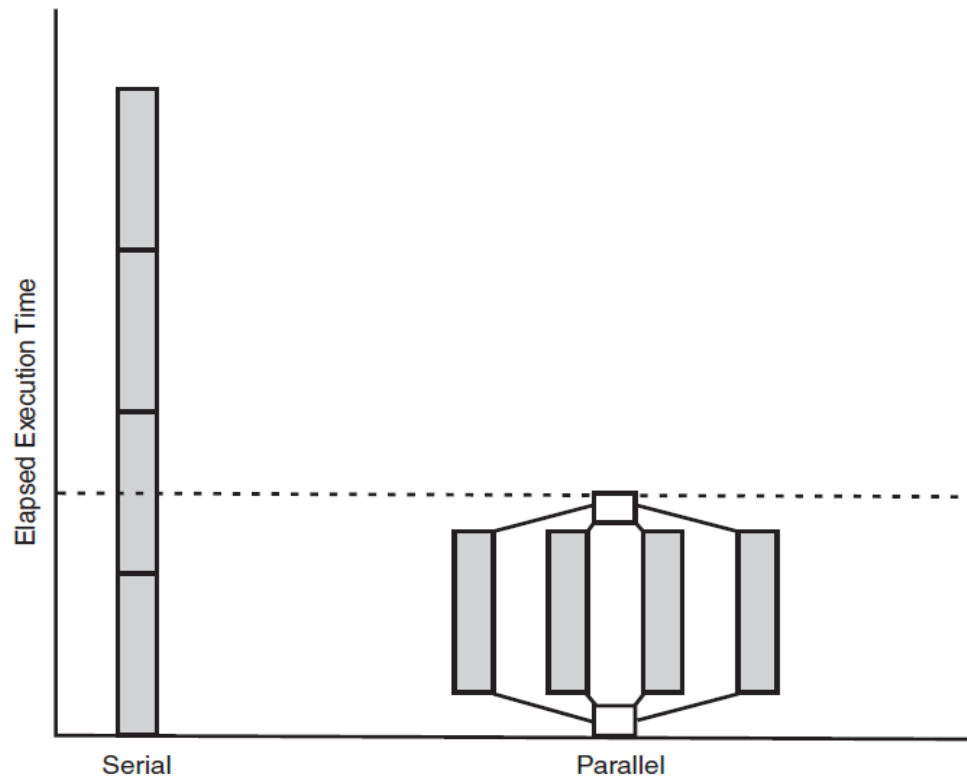
This is the subject for an entire presentation of its own; some advice

- Learn the access paths (described earlier) and how they are specified in the plan tables
- Use a visual explain tool to make access path reviews easier
 - IBM Data Studio
 - IBM Data Server Manager
 - Other vendor tools



Db2 for z/OS

- When you bind your program you can choose to enable query parallelism.
- Multiple parallel tasks used to access the data.



Parallelism in Db2 for LUW

- ◎ Db2 for LUW supports symmetric multiprocessor (SMP) machines.
 - ◎ This means more than one processor can access the database, allowing the execution of complex SQL requests to be divided among the processors.
 - ◎ This division of a single database operation into multiple tasks run in parallel within a single database partition is called *intrapartition parallelism*.
- ◎ From an application perspective, specifying parallelism is accomplished much the same as with Db2 for z/OS, using the DEGREE bind option or the CURRENT DEGREE special register.

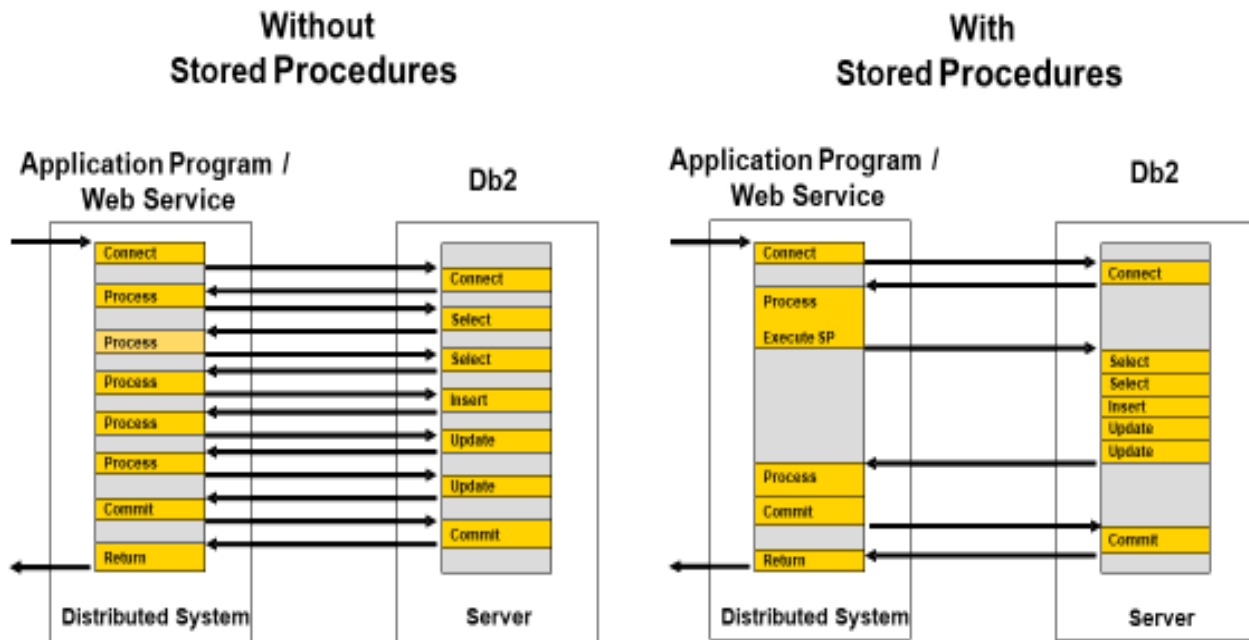
Types of Queries That Can Benefit from Parallelism

- ◎ The best candidates for parallel processing are I/O-bound queries.
- ◎ For example, the following queries are potential candidates for query I/O parallelism:
 - ◎ Queries accessing large amounts of data but returning only a few rows;
 - ◎ Queries using column functions (e.g. AVG, COUNT, MIN, MAX, SUM);
 - ◎ Queries against tables having long rows.

Chapter 19: Stored Procedures

- ⊙ There are six high-level use cases where stored procedures might be useful for your project:
 - ⊙ Reusability
 - ⊙ Consistency
 - ⊙ Data integrity
 - ⊙ Maintenance
 - ⊙ Performance, and
 - ⊙ Security

Stored Procedures can Reduce Network Calls



Avoid One-Size-Fits-All Implementations

- ◎ Be careful about implementing an application that relies too heavily on stored procedures. Sometimes “*advice*” is offered by well-meaning architects to use stored procedures for all database access.
 - ◎ For web-exposed applications, stored procedures can protect against SQL injection attacks.
 - ◎ Additionally, for developers using an ORM (object/relational mapping) tool such as Hibernate, stored procedures can be beneficial to avoid the inefficient SQL code typically generated by these tools.
 - If neither of these two situations apply, make sure that another compelling reason exists before requiring that all database access is accomplished using only stored procedures.

Chapter 20: Dynamic vs Static SQL

Dynamic SQL

- ⦿ Dynamic SQL is coded and embedded into an application program differently than static SQL. Not hard-coded, SQL is built “on the fly” as it executes.
- ⦿ Dynamic SQL statements must be compiled using the PREPARE statement; or, alternately, an implicit PREPARE is issued behind the scenes when implementing the EXECUTE IMMEDIATE flavor of dynamic SQL.

Static SQL

- ⦿ Static SQL is hard-coded and embedded into an application program. The SQL is bound into a package, which determines the access path that DB2 will use when the program is run.
- ⦿ Although dynamic SQL is more flexible than static, static SQL offers some flexibility by using host variables.

Static vs. Dynamic SQL Considerations

When to favor dynamic over static SQL:

- ⦿ Performance sensitivity of the SQL statement
 - ⦿ Dynamic SQL will incur a higher initial cost per SQL statement due to the need to prepare the SQL before use. But once prepared, the difference in execution time for dynamic SQL compared to static SQL diminishes.
- ⦿ Data uniformity
 - ⦿ Dynamic SQL can be more efficient than static SQL whenever data is:
 - Non-uniformly distributed. (e.g. cigar smokers skews male)
 - Correlated (e.g. CITY, STATE, and ZIP_CODE data will be correlated)
 - ⦿ Use of range predicates
 - The more frequently you need to use range predicates (<, >, <=, >=, BETWEEN, LIKE) the more you should favor dynamic SQL.
 - The optimizer can take advantage of distribution statistics & histogram statistics to formulate better access paths because the actual range will be known.

Static vs. Dynamic SQL Considerations (continued)

⦿ Repetitious Execution

- ⦿ As the frequency of execution increases, favor static SQL
 - Or perhaps dynamic SQL with local dynamic statement caching (KEEPDYNAMIC YES).
- ⦿ The cost of PREPARE becomes a smaller percentage of the overall run time of the statement the more frequently it runs (if cached prepare is reused).

⦿ Nature of Query

- ⦿ When you need all or part of the SQL statement to be generated during application execution favor dynamic over static SQL.

⦿ Run Time Environment

- ⦿ When the database objects may not exist at precompile time, dynamic SQL might be a better option than static specifying VALIDATE(RUN).

⦿ Frequency of RUNSTATS

- ⦿ When your application needs to access data that changes frequently and dramatically, it makes sense to consider dynamic SQL.

Chapter 21: Writing SQL – Many Ways to Skin a Cat

- ◎ SQL is a flexible language and you can code a query multiple different, compatible ways to end up with the same, correct answer
- ◎ Try multiple different formulations and test them for performance
 - ◎ Instead of just stopping when you get one that works
- ◎ Example: LIKE vs. OR vs. IN

```
SELECT Empno, Lastname, Sex
FROM   Emp
WHERE  Workdept LIKE 'A%';
```

```
SELECT Empno, Lastname, Sex
FROM   Emp
WHERE  Workdept = 'A00'
      OR Workdept = 'A01'
      OR Workdept = 'A02'...
```

```
SELECT Empno, Lastname, Sex
FROM   Emp
WHERE  Workdept IN ('A00', 'A01', 'A02'...);
```

Another Example

```
SELECT ColA, ColB, Col6
FROM Table1
WHERE Col1 NOT BETWEEN 'A' AND 'G';
```

non-indexable

```
SELECT ColA, ColB Col6
FROM Table1
WHERE COL1 >= 'H';
```

```
SELECT ColA, ColB, Col6
FROM Table1
WHERE Col1 BETWEEN 'H' AND 'Z';
```

**Of course, we
don't have
time to go
through all the
chapters...**

- ◎ Chapter 22: SQL Tips and Techniques for Performance
- ◎ Chapter 23: Tweaking SQL
- ◎ Chapter 24: Using Optimization Hints
- ◎ Chapter 25: Working with Views
 - ◎ Materialization vs. Merge
- ◎ Chapter 26: More Complex SQL Considerations
 - ◎ Temporal SQL
 - ◎ Recursive SQL
- ◎ Chapter 27: Testing Considerations
- ◎ Chapter 28: DevOps, Agile and Continuous Delivery
- ◎ Chapter 29: Resources

Contact Information



Craig S. Mullins
Mullins Consulting, Inc.

15 Coventry Ct
Sugar Land, TX 77479
craig@craigsmullins.com
www.mullinsconsulting.com



Use code **db2N** for 10% off print edition of book
Use code **db2W** for 5% off ebook edition of book
Expires 6/30/2020

A Guide to
Db2
Performance for
Application Developers

Code for performance
from the beginning

Craig S. Mullins

<https://tinyurl.com/db2-craig>