BP08: Art and Science of SQL Performance Tuning

Jarek Miszczyk
PartnerWorld for Developers, eServer iSeries

IBM eServer J iSeries J

ITSO iSeries Technical Forum

Science

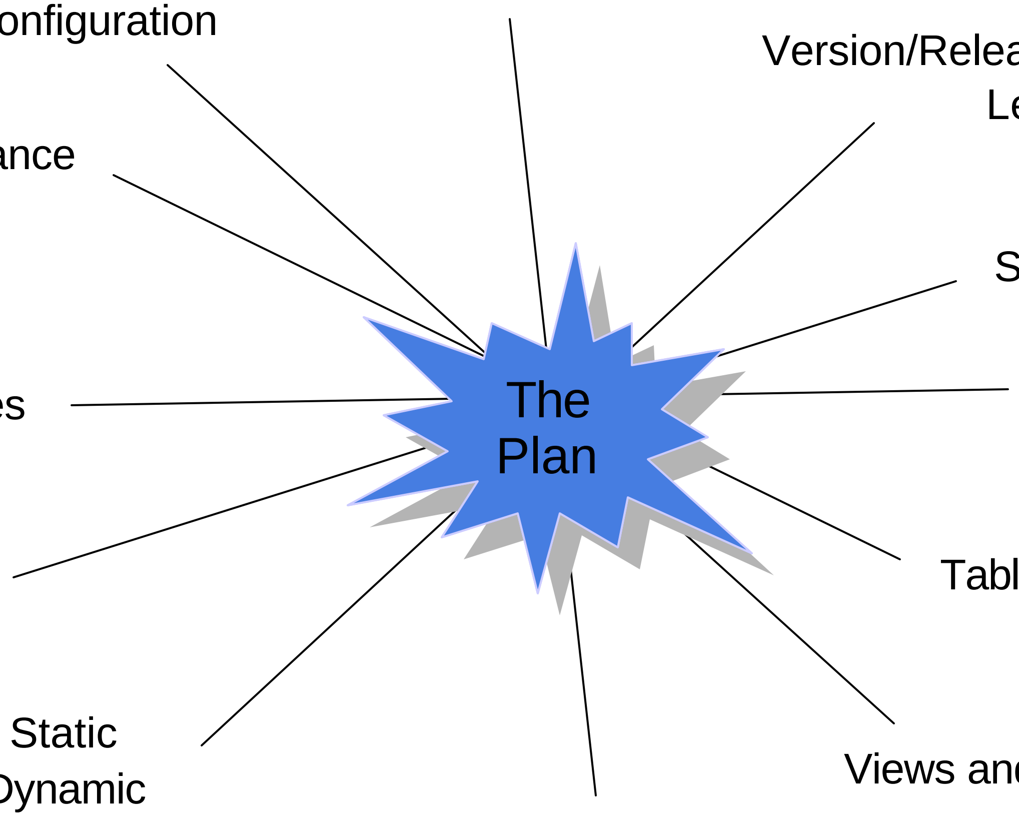
- What are you asking the system to do?
  - Type of request
  - SQL coding
- How can the system do it?
  - SQL Query implementation via the Optimizer, OS and SLIC
  - DB design
- Where is the system going to do it?
  - I/O Intensive
  - CPU Intensive
  - Available Resources

Optimization

- The Optimizer
  - Writes the best? program to fulfill your request

- The Optimizer
  - Provides the recipe
  - Provides the methods
  - Does no cooking

Optimization... the intersection of various factors



The Plan

Server attributes

Server configuration

Version/Release/Modification Level

Server performance

SMP

Job, Query attributes

Database design

SQL Request

Table sizes, number of rows

Static
Dynamic
Extended Dynamic
Interfaces

Views and Indexes (Radix, EVI)

Work management

Common Terms

Term	Meaning	Object
Table	Data repository or dataspace. Physical File.	*File, PF
Index	Binary radix tree built over a table to order particular columns (keys) of the table, useful for quick binary searches. Encoded vector (EVI) built over a table, used to create bitmaps for query processing. Logical File.	*File, LF
Temporary Index	Radix index built "on the fly" by the optimizer.	
Temporary Result	Copy of data from an intermediate step of the query. Needed to complete the query.	
Access Plan	Plan generated by the optimizer of how to access the tables being queried.	Dynamic, *PGM, *SRVPGM, *SQLPKG
ODP (Open Data Path)	Active path through which query data is read.	
Reusable ODP	ODP kept open by the system when an SQL query is repeatedly executed (run).	

Access Plans

Contents

- A control structure that contains information on the actions necessary to satisfy each SQL request
- These contents include:
  - Access Method
  - Info on associated tables and indexes
  - Any applicable program and/or environment information

Data Access

- Write a program to find the rows that contain the color purple within a 1 million row DB table

...WHERE COLOR = 'Purple'...

- When...
  - 1 row contains the color purple
  - 1,000 rows contain the color purple
  - 100,000 rows contain the color purple
  - 1,000,000 rows contain the color purple

Data Access

- SQL to find the rows that contain the color purple, within a 1 million row DB table, when...
  - 300,000 rows contain the color purple

```
SELECT ORDER, COLOR, QUANTITY
FROM ITEM_TABLE
WHERE COLOR = 'PURPLE'
```

- Without index over COLOR, assume 100,000 rows (10% default from =)
- With radix index over COLOR, estimate 291,357 rows (read keys)
- With EVI over COLOR, actual 300,000 rows (read symbol table)

Optimizer uses number of rows, not a percentage

Science - Implementation Methods

Implementation Methods Overview

- Non-Keyed Data Access Methods
  - Table Scan
  - Parallel Table Scan
  - Parallel Pre-fetch
  - Parallel Table Pre-load
  - Skip Sequential with dynamic bitmap
  - Parallel Skip Sequential
- Keyed Data Access Methods
  - Key Positioning and Parallel Key Positioning
  - Dynamic Bitmaps / Index ANDing ORing
  - Key Selection and Parallel Key Selection
  - Index-From-Index
  - Index-Only Access
  - Parallel Index Pre-load
- Joining, Grouping, Ordering
  - Nested Loop Join
  - Hash Join
  - Index Grouping
  - Hash Grouping
  - Index Ordering
  - Sort

Table Scan

Reads all rows from the table and applies the selection criteria to the data within the table.

- Advantages:
  - Minimizes page I/O operations through asynchronous pre-fetching of the data since the pages are scanned sequentially
  - Can perform selection directly on the table image in memory or on the intermediate buffer after all derived operations have been performed
- Potential disadvantages:
  - All rows in the table are examined regardless of the selectivity of the query
  - Rows marked as deleted are examined even though none will be selected
- Used when:
  - Greater than ~20% of the rows are selected
  - Table size is less than 32K

Table Scan Example

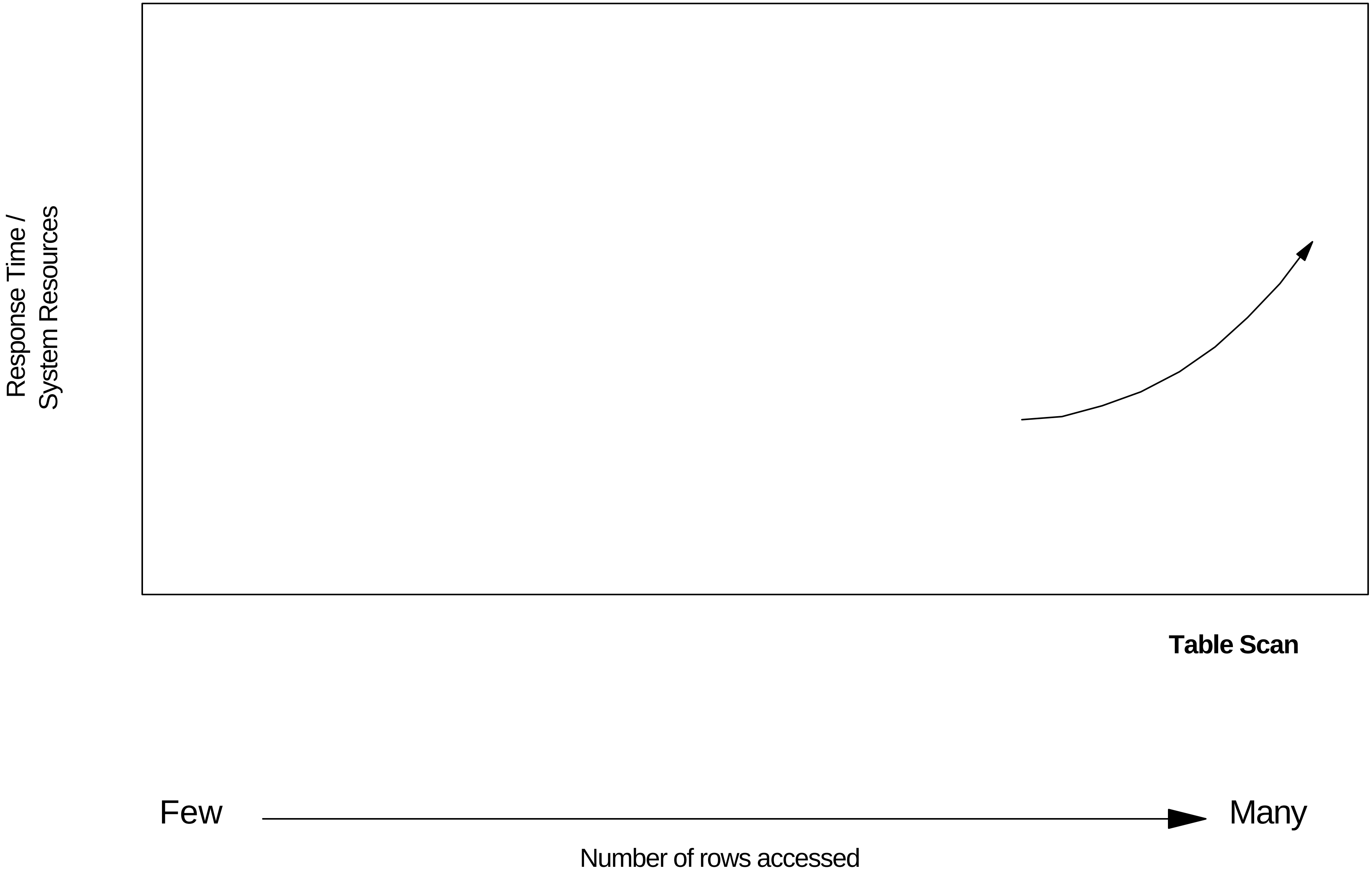
```
SELECT * FROM EMPLOYEE
   WHERE WORKDEPT BETWEEN 'A01' AND 'E01'
```

SQL4010 Arrival sequence access for file 1.

This message = Table scan

Why might a table scan be used...?

Data Access Methods



Encoded Vector Index (EVI)

- New index object for delivering fast data access in decision support and query reporting environments
  - Complementary alternative to existing index object (binary radix tree structure - logical file or SQL index)
  - Advanced technology from IBM Research, that's a variation on bitmap indexing
  - Easy to access data statistics improve query optimizer decision making

Encoded Vector Index (EVI)

What is it?

- New type of index
- File object type, LF subtype

EVI composed of two parts:

SYMBOL TABLE:

Key Value	Code	First Row	Last Row	Count
Arizona	1	1	80005	5000
Arkansas	2	5	99760	7300
......				
Virginia	37	1222	30111	340
Wyoming	38	7	83000	2760

VECTOR:

	Record
Code 1	1
Code 17	2
Code 18	3
Code 9	4
Code 2	5
Code 7	6
Code 38	7
Code 38	8
Code 1	9
...	...

- Symbol table contains information for each distinct key value. Each key value is assigned a unique hex code
  - Code is 1, 2, or 4 bytes - depending on number of distinct key values
- Rather than a bit array for each distinct key value, the index has one array of codes (a.k.a., the Vector)

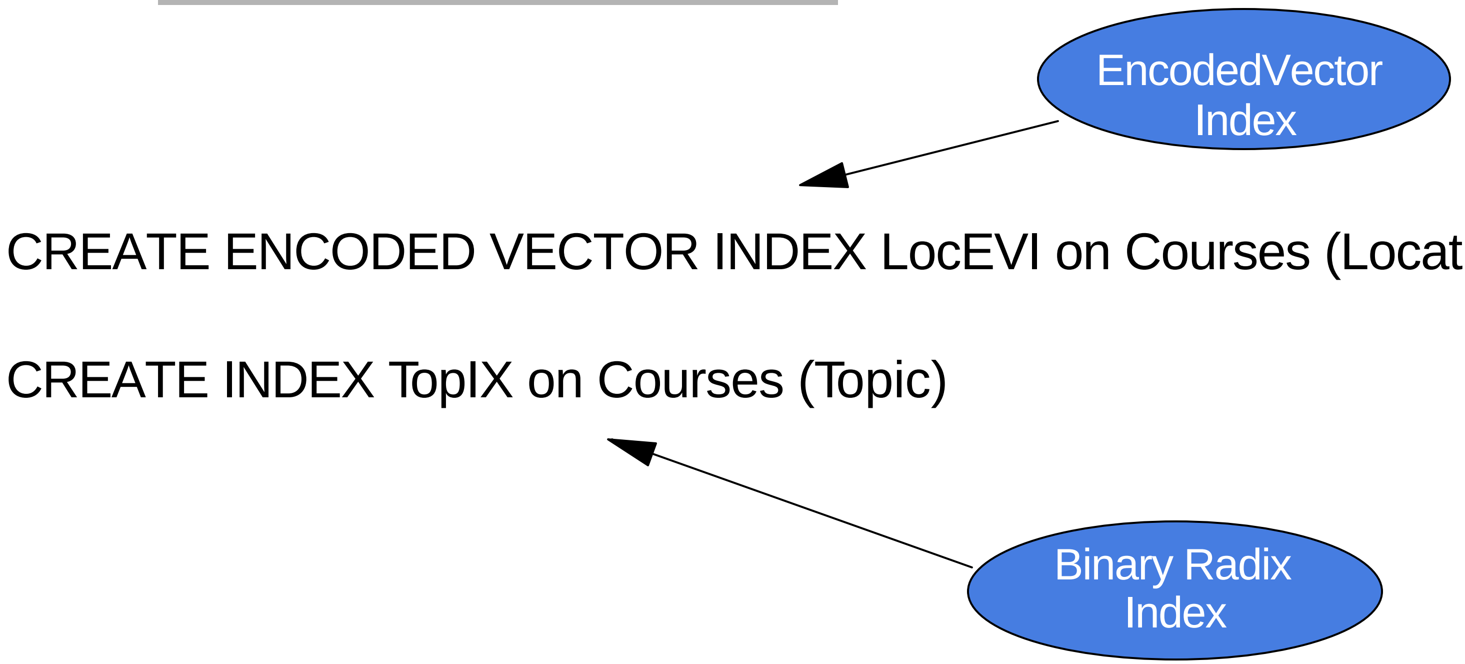
Dynamic Bitmaps

Index ANDing ORing

Courses

Location	Topic
NY	DB2/400
LA	JAVA
NY	JAVA
CHI	NOTES

Index LocEVI, created over the Location column and index TopIX, created over the Topic column in the Courses table.



EncodedVector Index

CREATE ENCODED VECTOR INDEX LocEVI on Courses (Location)

CREATE INDEX TopIX on Courses (Topic)

Binary Radix Index

Dynamic Bitmaps

Index ANDing ORing

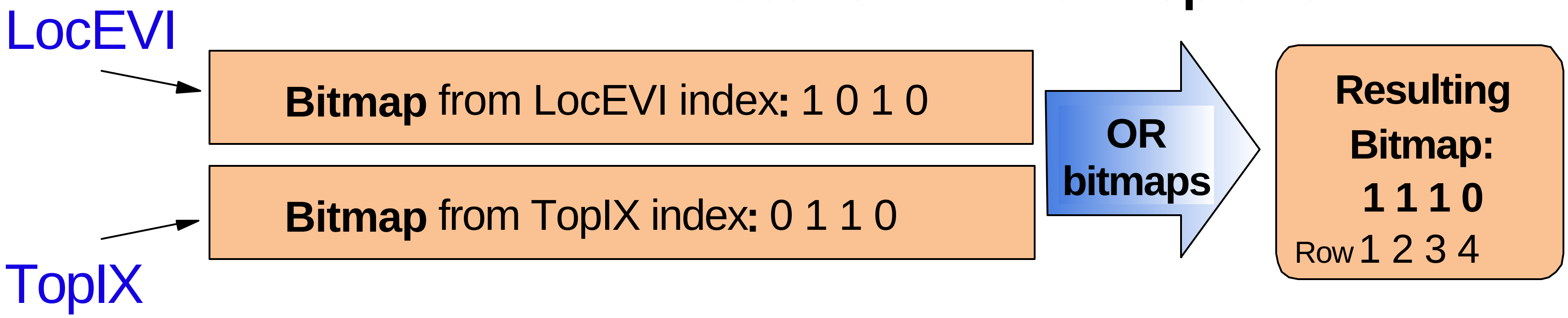
Courses

Location	Topic
NY	DB2/400
LA	JAVA
NY	JAVA
CHI	NOTES

Index LocEVI, has been created over the Location column and index TopIX, created over the Topic column in the Courses table.

SELECT coursenumber FROM courses

WHERE location='NY' OR topic='JAVA'



LocEVI

Bitmap from LocEVI index**:** 1 0 1 0

TopIX

Bitmap from TopIX index**:** 0 1 1 0

OR bitmaps

Resulting Bitmap:

1 1 1 0

Row 1 2 3 4

- Bitmaps can be derived from binary radix or encoded vector indices
  - bit order mirrors physical ordering of table data

Dynamic Bitmaps

Index ANDing ORing

Bitmaps are dynamically generated from existing indexes to reduce the I/O operations against the table

- Advantages:
  - Multiple indexes can be used against a single table
  - OR'ed predicates can be implemented with a tertiary index
  - Bitmaps can be generated and analyzed (logical AND and OR operations) in parallel
  - Can help to avoid some index creations
- Potential disadvantages:
  - The entire bitmap must be generated prior to retrieving any records
  - The generated bitmaps are static for the duration of the query
- Used when:
  - The savings from eliminating I/O operations outweigh the cost to generate and analyze the bitmap(s)

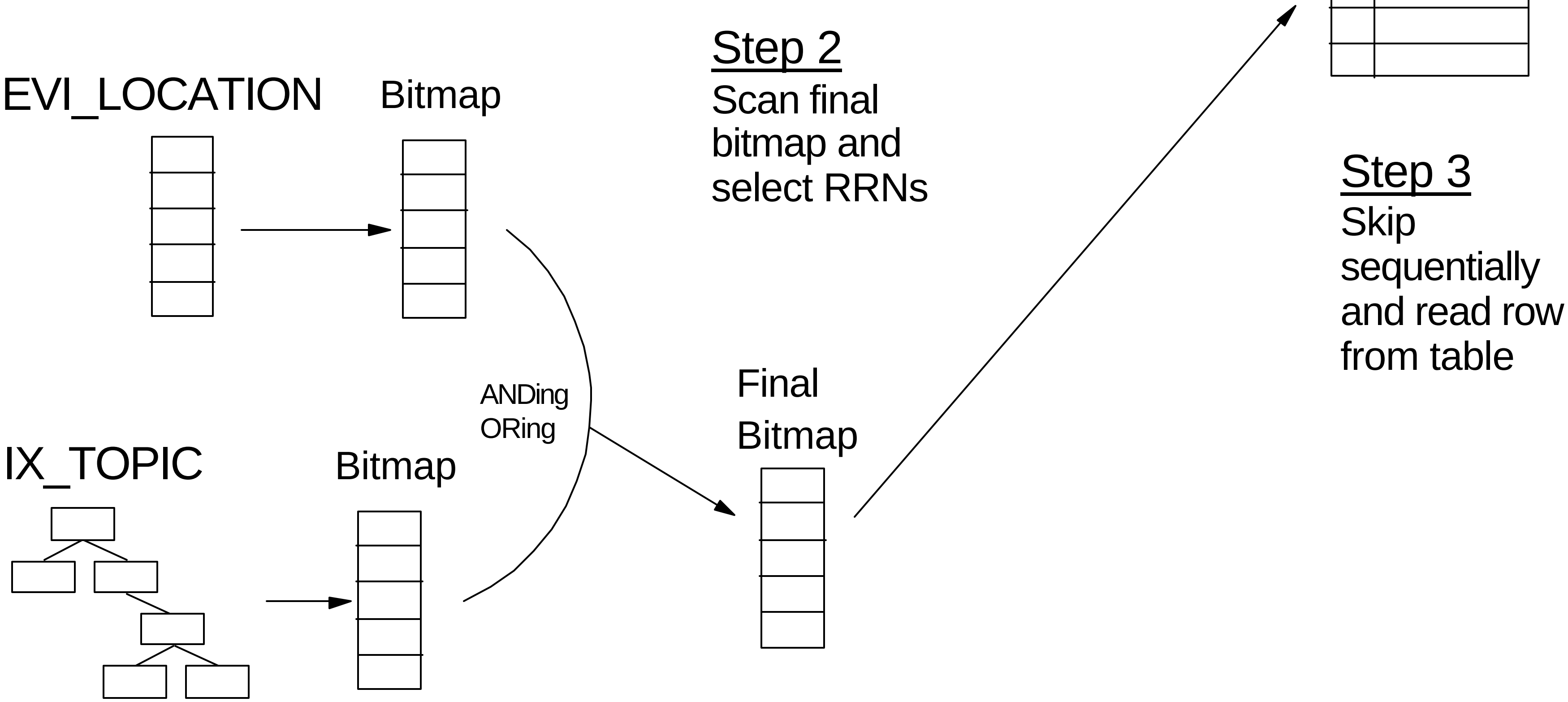
CPU parallelism

Skip Sequential Access

Scans the bitmap and sequentially reads rows from the table that match the selection criteria represented in the bitmap

- Advantages:
  - Minimizes page I/O operations through the use of dynamically generated bitmap(s), by skipping pages that have no rows represented in the bitmap
  - Minimizes page I/O operations through asynchronous pre-fetching of the data since the pages are scanned sequentially
  - Can perform selection directly on the table image in memory or on the intermediate buffer after all derived operations have been performed
- Potential disadvantages:
  - The entire bitmap must be generated prior to retrieving any records
  - The generated bitmaps are static for the duration of the query
- Used when:
  - Greater than ~20% of the rows are selected
  - The savings from eliminating I/O operations outweigh the cost to generate and analyze the bitmap(s)

Skip Sequential Access



Step 1
Select keys
and build dynamic
bitmaps

Step 2
Scan final
bitmap and
select RRNs

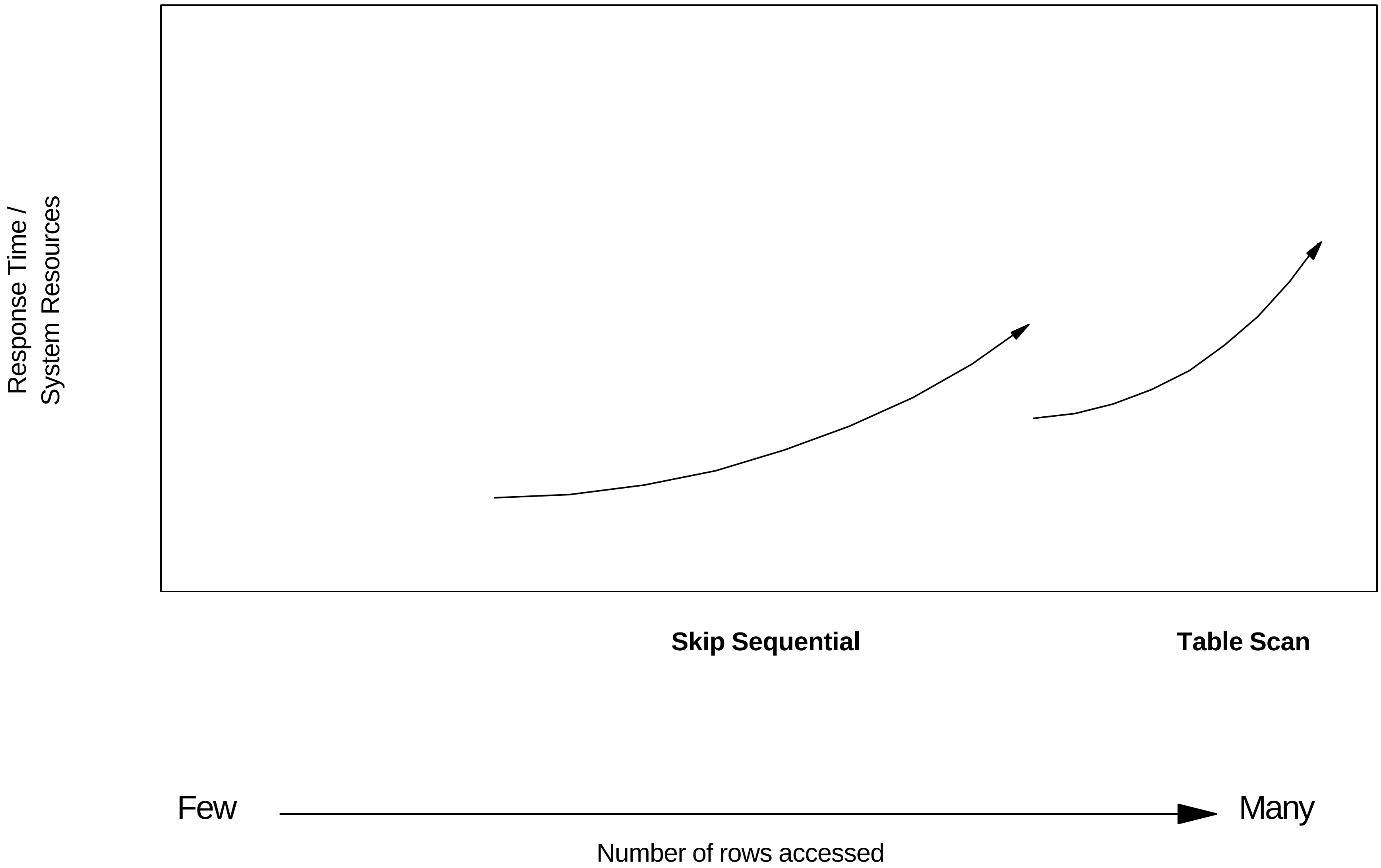
Step 3
Skip
sequentially
and read row
from table

Skip Sequential Example

```
CREATE ENCODED VECTOR INDEX EVI1 ON EMPLOYEE
   (WORKDEPT)
   :
SELECT * FROM EMPLOYEE
   WHERE WORKDEPT BETWEEN 'A01' AND 'E01'
```

```
SQL4010  Arrival sequence access for file 1.
SQL4032  Access path EVI1 used for bitmap processing of file 1.
```

Data Access Methods



Implementation Methods Overview

- Non-Keyed Data Access Methods
  - Table Scan
  - Parallel Table Scan
  - Parallel Pre-fetch
  - Parallel Table Pre-load
  - Skip Sequential with dynamic bitmap
  - Parallel Skip Sequential
- Keyed Data Access Methods
  - Key Positioning and Parallel Key Positioning
  - Dynamic Bitmaps / Index ANDing ORing
  - Key Selection and Parallel Key Selection
  - Index-From-Index
  - Index-Only Access
  - Parallel Index Pre-load
- Joining, Grouping, Ordering
  - Nested Loop Join
  - Hash Join
  - Index Grouping
  - Hash Grouping
  - Index Ordering
  - Sort

Key Positioning

Selection criteria are applied to ranges of index entries before the table is processed.

- Advantages:
  - Only those index entries that are within a selected range are processed
  - Can process both join and selection processing within a single operation if the correct index exists
- Potential disadvantages:
  - Can perform poorly when a large number of rows are selected
- Used when:
  - Less than ~20% of the keys are selected
  - Ordering, grouping, or join operation requires the use of an index
  - The selection columns match the first (n) key fields of the index
  - May be used in combination with key selection

Key Positioning Example

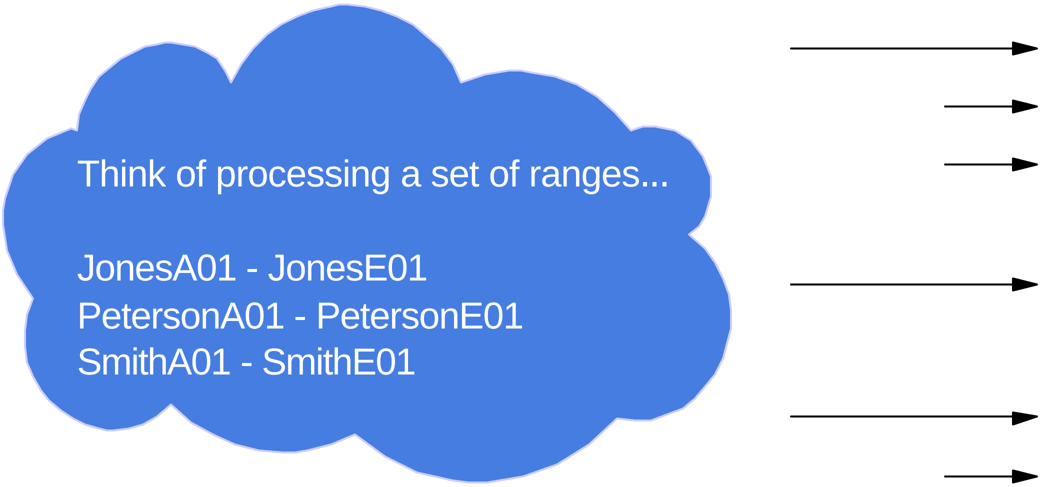
```
CREATE INDEX X1 ON EMPLOYEE
   (LASTNAME, WORKDEPT)
   :
SELECT * FROM EMPLOYEE
   WHERE WORKDEPT BETWEEN 'A01' AND 'E01'
      AND LASTNAME IN ('SMITH', 'JONES', 'PETERSON')
```

SQL4008 Access path X1 used for file 1.

SQL4011 Key row positioning used on file 1.

Key Positioning Example

```
CREATE INDEX X1 ON EMPLOYEE
     (LASTNAME, WORKDEPT)
     :
SELECT * FROM EMPLOYEE
     WHERE WORKDEPT BETWEEN 'A01' AND 'E01'
          AND LASTNAME IN ('SMITH', 'JONES', 'PETERSON')
```



Think of processing a set of ranges...

JonesA01 - JonesE01
PetersonA01 - PetersonE01
SmithA01 - SmithE01

LASTNAME	WORKDEPT
Adamson	B01
Anderson	B01
Anderson	G01
Cain	A01
Caine	G01
Doe	E01
Jones	A01
Jones	C01
Jones	D01
Milligan	A01
Peterson	C01
Peterson	F01
Smith	B01
Smith	C01
Smith	D01
Smith	F01
Wulf	A01

Key Selection

Selection criteria are applied to the key(s) of the index before the table page is retrieved.

- Advantages:
  - The table is only accessed for rows that satisfy the key selection criteria
- Potential disadvantages:
  - The entire index is read and the key selection criteria is applied to each key entry
  - A random I/O is performed against the table for each key selected from the index
  - Can perform poorly when a large number of rows are selected
- Used when:
  - Less than ~20% of the keys are selected
  - Ordering, grouping, or join operation requires the use of an index
  - May be used in combination with key positioning

Key Selection Example

```
CREATE INDEX X1 ON EMPLOYEE
    (LASTNAME, WORKDEPT)
    :
SELECT * FROM EMPLOYEE
    WHERE WORKDEPT BETWEEN 'A01' AND 'E01'
```

SQL4008 Access path X1 used for file 1.

Key Selection Example

```
CREATE INDEX X1 ON EMPLOYEE
     (LASTNAME, WORKDEPT)
     :
SELECT * FROM EMPLOYEE
     WHERE WORKDEPT BETWEEN 'A01' AND 'E01'
```

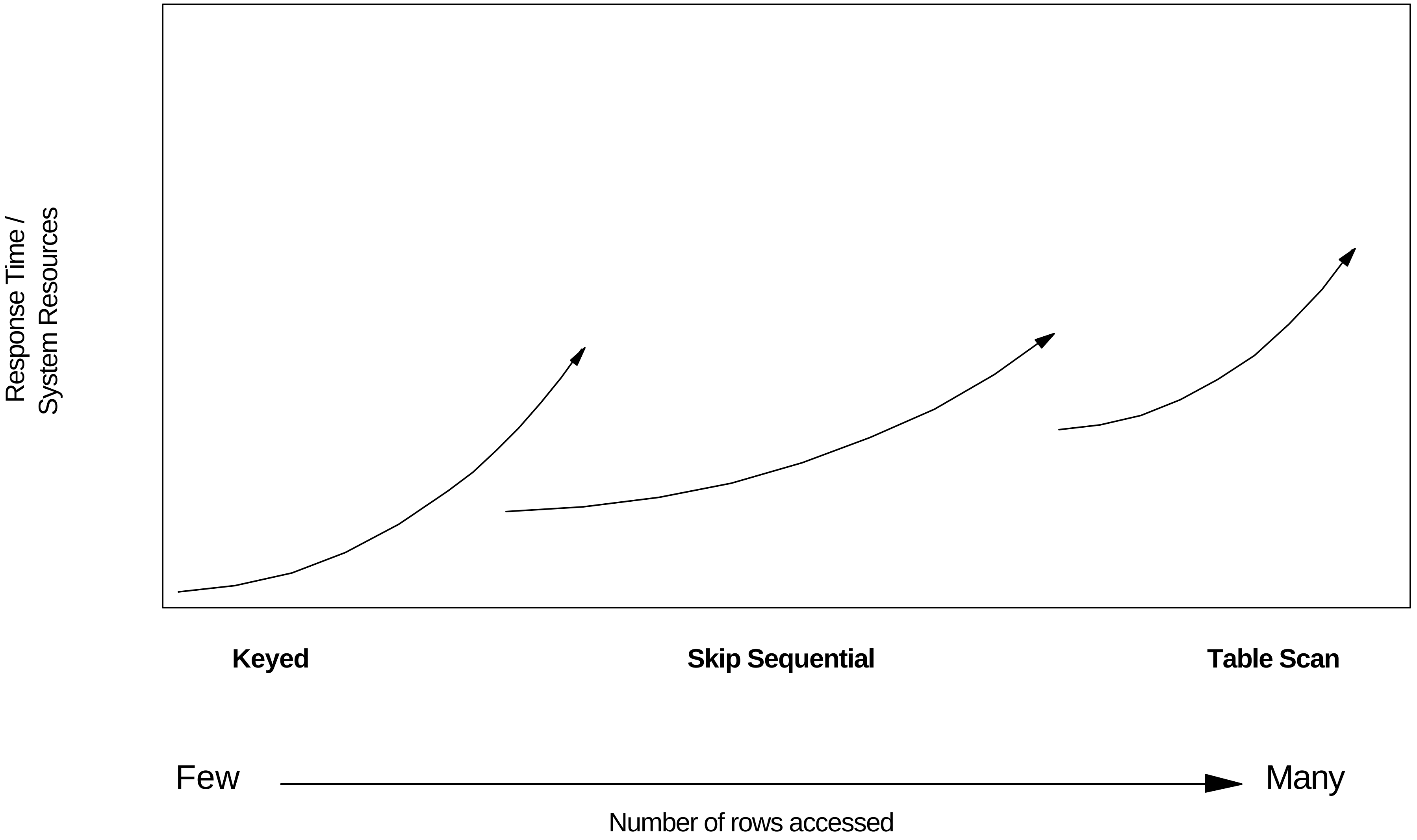
Think of scanning the entire index...

testing WORKDEPT for A01 - E01

LASTNAME	WORKDEPT	
Adamson	B01	←
Anderson	B01	←
Anderson	G01	
Cain	A01	←
Caine	G01	
Doe	E01	←
Jones	A01	←
Jones	C01	←
Jones	D01	←
Milligan	A01	←
Peterson	C01	←
Peterson	F01	
Smith	B01	←
Smith	C01	←
Smith	D01	←
Smith	F01	
Wulf	A01	←

...

Data Access Methods



Implementation Methods Overview

- Non-Keyed Data Access Methods
  - Table Scan
  - Parallel Table Scan
  - Parallel Pre-fetch
  - Parallel Table Pre-load
  - Skip Sequential with dynamic bitmap
  - Parallel Skip Sequential
- Keyed Data Access Methods
  - Key Positioning and Parallel Key Positioning
  - Dynamic Bitmaps / Index ANDing ORing
  - Key Selection and Parallel Key Selection
  - Index-From-Index
  - Index-Only Access
  - Parallel Index Pre-load
- Joining, Grouping, Ordering
  - Nested Loop Join
  - Hash Join
  - Index Grouping
  - Hash Grouping
  - Index Ordering
  - Sort

Joins
Common Terms

Term	Meaning
Join Position	Position in which this file is being joined.
Join Dial	Same as Join Position.
Join Order	The order of all of the files used to process the join. (Dial1 --> Dial2 --> Dial3 --> Dial4)
Average Duplicates	Average number of records for each distinct value. Statistic derived from an index.
Dial	Synonymous with the odometer on a vehicle. For each record of dial 1, you must spin through all of the records of dial 2.
Join Fanout	The number of join combinations that can be expected for each join value.

Join Support for SQL

- For inner join, optimizer not biased toward using specified join order
- For left outer and exception join, tables are joined from left to right
  - INNER JOIN tables can be reordered
- Multiple join types supported for a single query
- Join implementation methods
  - Nested Loop
  - Hash

Nested Loop Joins

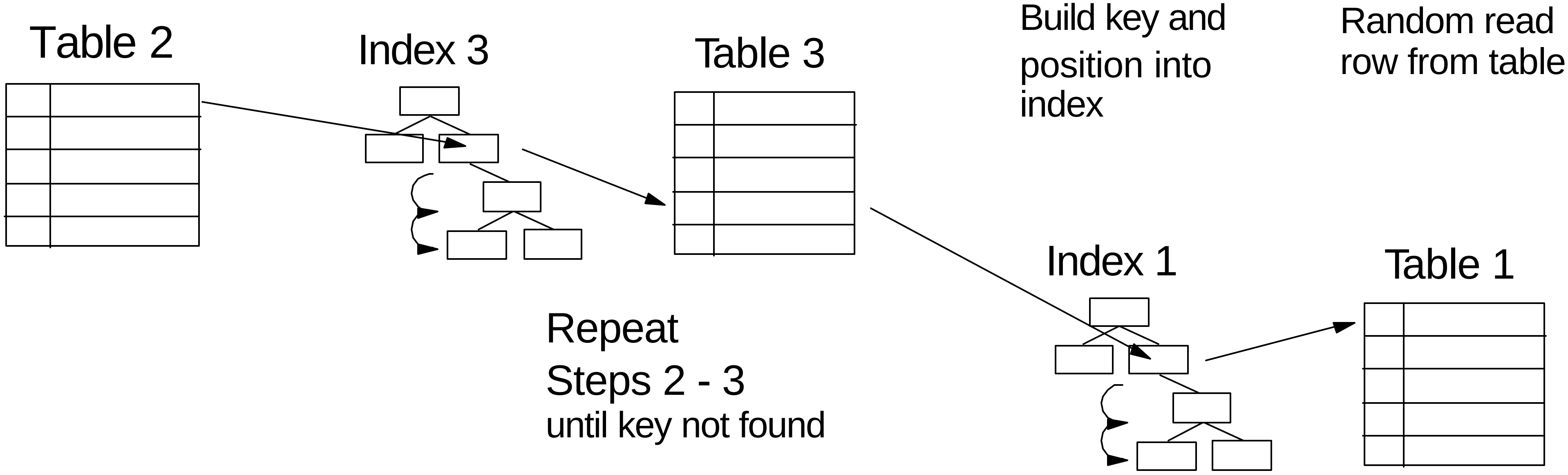
- Each row selected from the primary file is joined to each secondary file using a key value built over the join-to fields.
  - The join spins like an odometer on a car (from right to left).
  - After a file has been completely cycled, then it backs up to the previous dial and gets the next join value.
  - The join is performed again spinning through the next secondary file in the odometer (N-1).
- The join is not complete until all the rows of the primary file have been processed.

Nested Loop Joins

Step 1
Select row
and build key

Step 2
Position into
index

Step 3
Random read
row from table



Step 4
Build key and
position into
index

Step 5
Random read
row from table

Repeat
Steps 2 - 3
until key not found

```
SELECT * FROM TABLE_1, TABLE_2, TABLE_3
WHERE FKEY1 = PKEY3
AND FKEY2 = PKEY3
```

Repeat
Steps 4 - 5
until key not found

Nested Loop Joins

Creating a Temporary Index for the Join Criteria

- If an index over the join fields of the secondary file(s) does not exist, one is created.
- Advantages:
  - Since local selection is performed ahead of the join (during index creation) the temporary index generally is smaller so there are less index pages to be faulted in.
- Disadvantages:
  - Creating a temporary index is very CPU intensive and is not suitable for OLTP.
  - If index is built using host variable selection, then the query is not reusable.

Optimizer may create a temporary index when a permanent index exists. If the join fan out is high (or the cost of the NL join is very high) the optimizer may chose to create a sparse, temporary index, trying to make the join as efficient as possible.

Hashing Algorithm

Joining

A hashing algorithm is used to correlate data with a common value together for grouping and/or join queries.

- Advantages:
  - Allows for the exploitation of CPU parallelism for the creation of the hash table
  - Reduces the random I/O to the table generally associated with longer running queries using an index
  - Generally faster than creating a temporary index to perform the specified operation
- Potential disadvantages:
  - May perform poorly when processing a small subset of the rows that will be used as input into the hashing algorithm
  - A temporary copy of the data is required to process the hash
- Used when:
  - The value *OPTIMIZE is specified or *YES if a temporary file is required, on the ALWCPYDTA parameter
  - Grouping and/or a join processing is specified in the query

CPU parallelism

Hash Join

- The hash join uses a hash table to correlate all of the distinct join values into common buckets (hash points) and then using the buckets to find all of the join combinations.
- Query rewritten to take advantage of SMP
- This has the following improvements over a Nested Loop Join:
  - An index is no longer required to find the join matches.
  - No longer have to iterate through all of the files for each of the join possibilities (i.e. odometer processing).
- Equal join predicates only
- No specific information on hash joins in DB monitor
  - Query to select records and build hash table(s) do show up

Hash Join

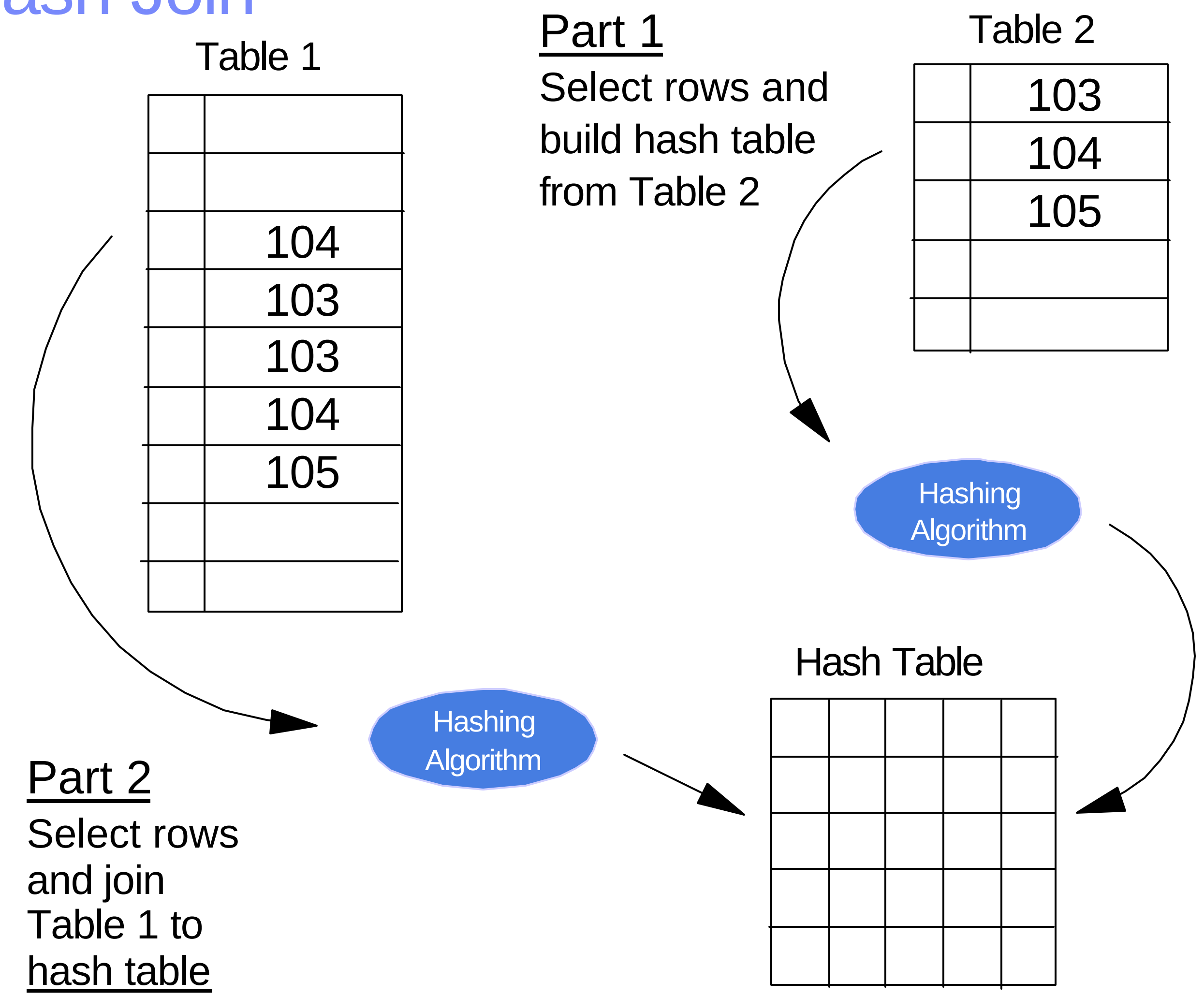


Table 1

	104
	103
	103
	104
	105

Part 1

Select rows and build hash table from Table 2

Table 2

	103
	104
	105

Part 2

Select rows and join Table 1 to hash table

Like join values are grouped together by their hash value and all the appropriate data is stored.
Collisions are handled by linked list.

Example:

SELECT *
FROM TABLE1 A, TABLE2 B
WHERE A.PARTKEY in ('103', '104', '105')
and A.PARTKEY = B.PARTKEY

Hash Join Example

```
SELECT * FROM EMPLOYEE, DEPARTMENT
   WHERE EMPLOYEE.WORKDEPT = DEPARTMENT.DEPTNO
      AND EMPLOYEE.HIREDATE BETWEEN '1995-01-30'  AND
         '1996-01-30'
      AND DEPARTMENT.DEPTNO IN ('A00', 'D01', 'D21', 'E11')
   OPTIMIZE FOR ALL ROWS
```

SQL402A Hashing algorithm used to process join.

SQL402B File EMPLOYEE used in hash join step 1.

SQL402B File DEPARTMENT used in hash join step 2.

Join Optimization

- The main optimization rule of thumb for a join query is the reordering of the files.
  - This minimizes the join fanout and that in turn minimizes I/Os.
- Reordering of files is allowed only on inner joins. Left Outer or Exception joins cannot be reordered.
- The DB2 UDB for iSeries optimizer uses a greedy join algorithm to determine the most efficient table order.

Greedy Join Algorithm

- Determine best access for each individual file, ignoring the join for a moment (keyed access, data space scan, etc.)
- For each join combination determine the join cost
  - For a 4 file join, the join combinations would be:

    1-2, 2-1, 1-3, 3-1, 1-4, 4-1, 2-3, 3-2, 2-4, 4-2, 3-4, 4-3
- The combination with lowest cost determines the primary and first secondary files

  2 3 x x
- For each remaining file, determine cost of joining to previous files
  - File with lowest cost becomes next secondary file

  2 3 1 x
- Repeat join cost calculation until complete join order has been determined

  2 3 1 4

Implementation Methods Overview

- Non-Keyed Data Access Methods
  - Table Scan
  - Parallel Table Scan
  - Parallel Pre-fetch
  - Parallel Table Pre-load
  - Skip Sequential with dynamic bitmap
  - Parallel Skip Sequential
- Keyed Data Access Methods
  - Key Positioning and Parallel Key Positioning
  - Dynamic Bitmaps / Index ANDing ORing
  - Key Selection and Parallel Key Selection
  - Index-From-Index
  - Index-Only Access
  - Parallel Index Pre-load
- Joining, Grouping, Ordering
  - Nested Loop Join
  - Hash Join
  - Index Grouping
  - Hash Grouping
  - Index Ordering
  - Sort

Group-By Optimization

- The query optimizer chooses between index grouping and hash grouping
- Indexes are used for grouping statistics (number of groups)
  - Keys over grouping column(s)
  - Average duplicates statistics
  - Number of hash points (hash table size)
    - 16K
    - 64K
- Query attributes affect which method is used
  - Index Group by
    - First I/O
    - ALWCPYDTA(*NO), ALWCPYDTA(*YES)
  - Hash Group by
    - All I/O
    - ALWCPYDTA(*OPTIMIZE)

Implementation Methods Overview

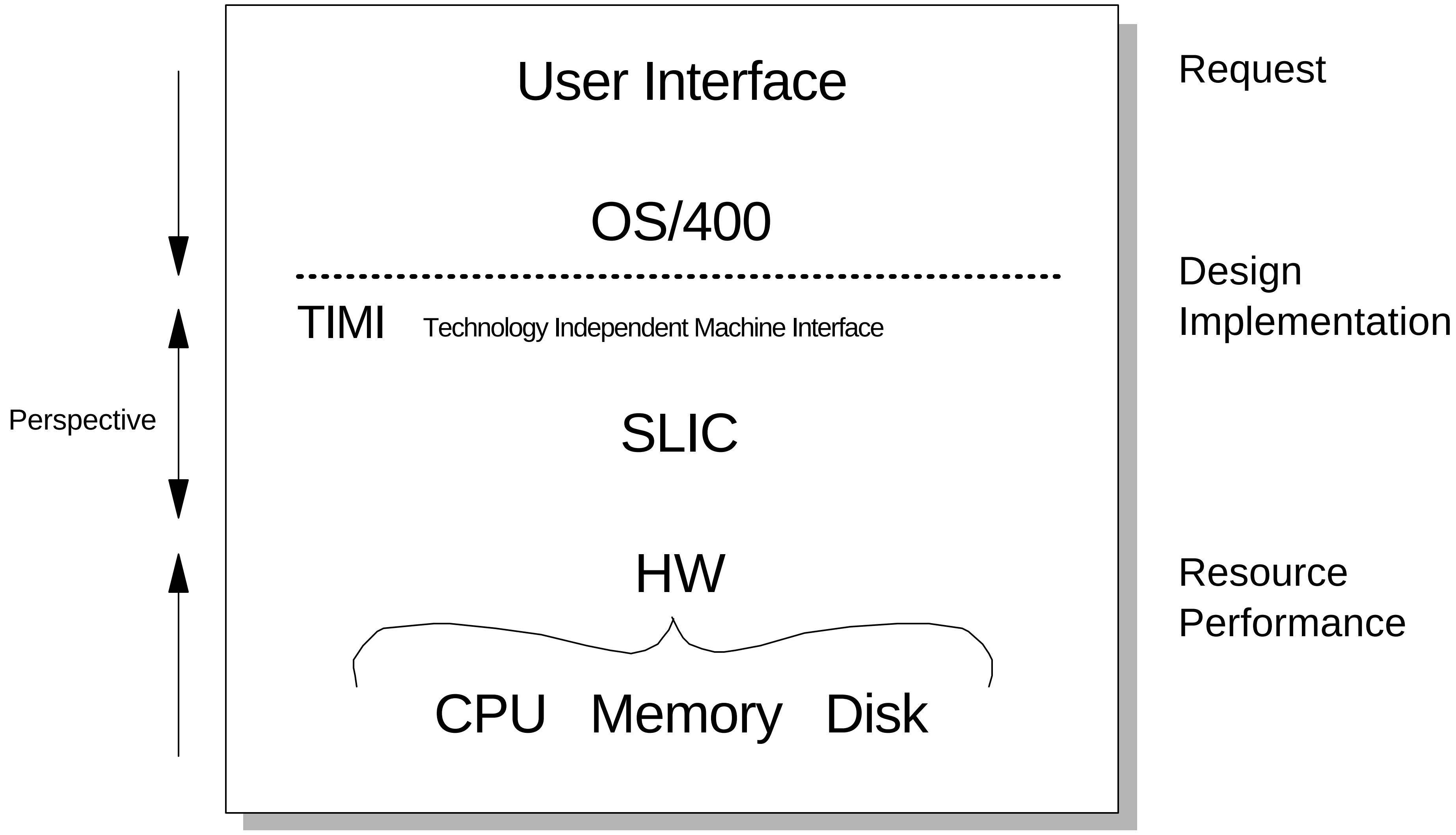
- Non-Keyed Data Access Methods
  - Table Scan
  - Parallel Table Scan
  - Parallel Pre-fetch
  - Parallel Table Pre-load
  - Skip Sequential with dynamic bitmap
  - Parallel Skip Sequential
- Keyed Data Access Methods
  - Key Positioning and Parallel Key Positioning
  - Dynamic Bitmaps / Index ANDing ORing
  - Key Selection and Parallel Key Selection
  - Index-From-Index
  - Index-Only Access
  - Parallel Index Pre-load
- Joining, Grouping, Ordering
  - Nested Loop Join
  - Hash Join
  - Index Grouping
  - Hash Grouping
  - Index Ordering ←
  - Sort

Order-By Optimization

- The query optimizer chooses between index ordering and sort
- Optimizer costs the use of each method and picks the fastest
- Query attributes affect which method is used
  - Index Order by
    - First I/O
    - ALWCPYDTA(*NO), ALWCPYDTA(*YES)
  - Sort Order by
    - All I/O
    - ALWCPYDTA(*OPTIMIZE)

Art of SQL Optimization

iSeries Architecture



What can be done...?

- Change the request or SQL coding
- Change the design or influence the implementation
  - Database design
  - Tuning "knobs" and indexes
  - Upgrade OS to obtain new features
- Change the resource performance
  - Work Management
  - Additional or upgraded hardware
  - SMP
- Change response time expectations

Art

- Environments
  - A few long running or complex requests
    - Dedicate all resources
    - SMP
    - Highly tuned
  - Many quick, small or medium ad-hoc requests
    - Share resources (like OLTP)
    - Little or no SMP
    - Unpredictable - no chance to tune
  - Mixture
    - Separate environments
    - Separate systems or logical partitions

Considering the entire request...

- SELECTION
  - Keyed access SMP
  - Skip Sequential access SMP
  - Sequential access SMP
- JOIN
  - Nested loop (via keyed access)
  - Hash SMP
- GROUP BY
  - Index
  - Hash SMP
- ORDER BY
  - Index
  - Sort

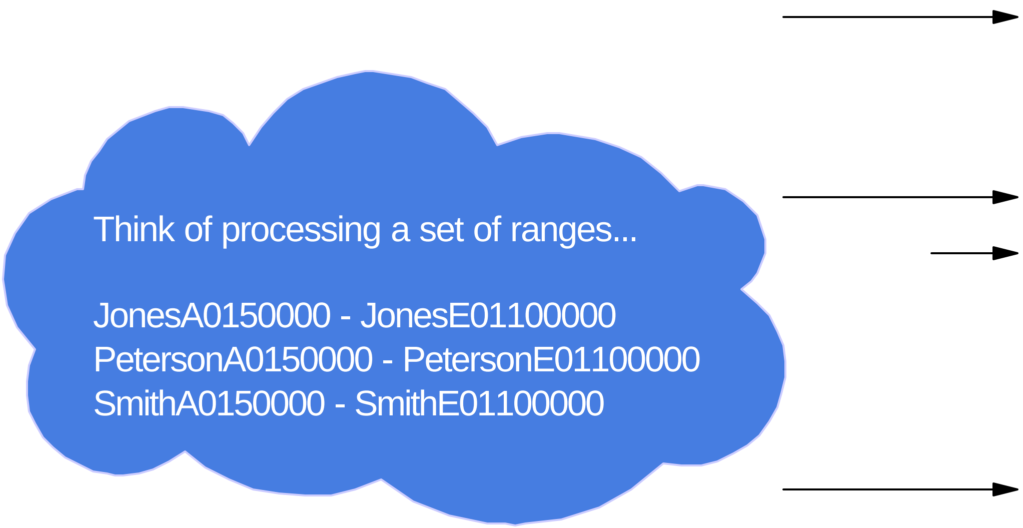
Art - The Perfect Index

- A "perfect" index is a radix index that is permanent and can provide:
  - Good, useful statistics to the optimizer
    - Index contains appropriate selection, joining, grouping, ordering fields
    - Applicable key fields are contiguous
    - Equal predicate fields first, one non-equal predicate field last
  - Multiple implementation methods
    - Index ANDing / ORing with dynamic bitmaps
    - Single key and multi-key row positioning
    - Index scan
    - Index only access
    - Nested loop join (with multi-key row positioning)
    - Index grouping
    - Index ordering
  - Multi-key index that provides very narrow range of values
    - Think in terms of lower and upper bounds

Art - The Perfect Index

```
CREATE INDEX X1 ON EMPLOYEE
     (LASTNAME, WORKDEPT, SALARY)
     :
SELECT * FROM EMPLOYEE
     WHERE WORKDEPT BETWEEN 'A01' AND 'E01'
          AND LASTNAME IN ('SMITH', 'JONES', 'PETERSON')
          AND SALARY BETWEEN 50000 AND 100000
```

LASTNAME	WORKDEPT	SALARY
...	...	
...	...	
...	...	
Jones	A01	35000
Jones	C01	51000
Jones	D01	45000
...	...	
Peterson	C01	60000
Peterson	E01	100000
Peterson	E01	120000
...	...	
Smith	B01	47000
Smith	C01	59000
Smith	F01	62000
...	...	
...	...	



Think of processing a set of ranges...

JonesA0150000 - JonesE01100000
PetersonA0150000 - PetersonE01100000
SmithA0150000 - SmithE01100000

Early elimination of rows is the key

- Narrow range(s)

"Perfect" radix indexes...

- SELECTION
  - Keyed access SMP
  - Skip Sequential access SMP
  - Sequential access SMP
- JOIN
  - Nested loop (via keyed access)
  - Hash SMP
- GROUP BY
  - Index
  - Hash SMP
- ORDER BY
  - Index
  - Sort

"Perfect" radix indexes...

Statistics
Multi key selection
Index only access
Nested loop join
Index grouping
Index ordering

Selection, grouping, ordering

```
CREATE INDEX IX1 on TABLE1 (YEAR, MONTH, CUSTOMER, ORDERNO)
```

Joining

```
CREATE INDEX IX2 on TABLE2 (ORDERNO, QUANTITY, SALES_AMOUNT)
```

```
SELECT A.YEAR, A.MONTH, A.CUSTOMER, SUM(B.QUANTITY),
SUM(B.SALES_AMOUNT)
    FROM TABLE1 A, TABLE2 B
    WHERE A.YEAR = 2000 and A.MONTH in (10, 11, 12) and A.CUSTOMER = 'SMITH'
    and A.ORDERNO = B.ORDERNO
    GROUP BY A.YEAR, A.MONTH, A.CUSTOMER
    ORDER BY A.YEAR, A.MONTH, A.CUSTOMER
```

Multiple techniques...

- SELECTION
  - Keyed access SMP
  - Skip Sequential access SMP
  - Sequential access SMP
- JOIN
  - Nested loop (via keyed access)
  - Hash SMP
- GROUP BY
  - Index
  - Hash SMP
- ORDER BY
  - Index
  - Sort

Multiple techniques...

Some statistics
Parallel skip sequential
Parallel table scan
Hash join
Hash grouping
Sort

Selection via
dynamic bitmap

```
CREATE ENCODED VECTOR INDEX IX1 on TABLE1 (YEAR, MONTH)


SELECT A.YEAR, A.MONTH, A.CUSTOMER, SUM(B.QUANTITY),
SUM(B.SALES_AMOUNT)
    FROM TABLE1 A, TABLE2 B
    WHERE A.YEAR = 2000 and A.MONTH in (10, 11, 12) and A.CUSTOMER = 'SMITH'
    and A.ORDERNO = B.ORDERNO
    GROUP BY A.YEAR, A.MONTH, A.CUSTOMER
    ORDER BY A.YEAR, A.MONTH, A.CUSTOMER
```

Tuning "knobs"...

- ALWCPYDTA
  - *NO
  - *YES
  - *OPTIMIZE
- OPTIMIZE FOR n ROWS, OPTIMIZE FOR ALL ROWS
  - *FIRSTIO
  - *ALLIO
- CHGQRYA / QQRYDEGREE
  - *NONE
  - *IO
  - *NBRTASKS
  - *OPTIMIZE
  - *MAX
- QAQQINI file

Indexing Strategies - Basic Approach

- You must create some indexes
  - Statistics
  - Implementation
- In general: equal selection columns first, then join columns -or- group-by and order-by columns
- Be aware of limitations when creating null capable key columns (no index only access)
- May have to play around with key order based on the queries, the data and selectivity of the columns
- Consider Index Only Access
  - All columns in the SELECT clause as keys
- Consider dynamic bitmaps and index ANDing/ORing
  - Simple indexes can be combined together for selection
- Consider EVIs for stats, dynamic bitmaps, and star schema join
  - Single key, low number of unique values
  - Fact table foreign key
  - Over temporary results file to provide stats
- Check for messages and iterate

Indexing Strategies cont.

- Run queries on system tables to find table and associated indexes
  - Example tool can be found at: www.iseries.ibm.com/db2/indexlist.htm
- Use ANZDBF and ANZDBFKEY commands to produce reports that show tables, indexes and key analysis
- Tables with truly unique keys, specify UNIQUE on the index or PK constraint
  - Primary key and RI constraints use "hidden" indexes that the optimizer can use
- Proactive
  - Create indexes over primary, foreign key columns and dependent columns
  - Create indexes for selection and joining
  - Create indexes for selection, grouping and ordering
- Reactive
  - Create indexes based on optimizer feedback
  - Create indexes based on optimization, implementation, system resources and performance

Indexing Strategies

- Proactive
  - Analyze the SQL request
- Reactive
  - Rely on feedback and actual implementation methods
- Profile long running or complex queries to capture the optimization and implementation method
- Profile the system when running long or complex queries to capture work management configurations and performance statistics
- Understand the data being queried
- Consider separating complex queries into two or more queries and tune individual parts
  - Join query + Group-by query
  - Subquery

Art - Tools and Methodologies

Overview

- How do I know what's going on with my queries?
- How can I tell what the optimizer is doing?
- Answer: Tools and analysis
  - Query optimizer debug messages
  - Print SQL Information (PRTSQLINF)
  - Database Monitor Statistics
    - Detailed Monitor (STRDBMON)
    - Summary (Memory-based) Monitor (Operations Navigator)
  - Visual Explain
  - Change Query Attributes (CHGQRYA)
  - QAQQINI file attributes

Debug Messages

- Informational messages written to the joblog about the implementation of a query
- Describes query implementation method
  - Indexes
  - Join order
  - Access plans
  - ODPs (Open Data Paths)
- Messages explain what happened during query optimization
  - Why index was or was not used
  - Why a temporary index result was required
  - Index advised by the optimizer
- STRDBG UPDPROD(*YES) & STRSRVJOB and STRDBG for batch jobs
- ODBC & JDBC Driver Exit program
- MESSAGES_DEBUG = *YES in QAQQINI file

Print SQL Information

- OS/400 command that lists SQL information contained in a program, SQL package, or service program.

```
PRTSQLINF OBJ(MY_PGM) OBJTYPE(*PGM)

PRTSQLINF OBJ(MY_PKG) OBJTYPE(*SQLPKG)
```

- Creates a spooled file that contains:
  - SQL statements
  - Type of access plan used by each statement
  - Command (CRTSQLxxx) and parameters used to invoke the SQL precompiler
- iSeries version of SQL EXPLAIN utility
- Output similar to debug messages

Database Monitors

- Integrated tools used to gather database performance related statistics for SQL-based requests
- Monitor data dumped into table(s) where it can be queried to help identify and tune performance problem areas
  - Detailed monitor writes all of the information out to a single table as it's collected
    - Interface: STRDBMON & Operations Navigator
  - Summary monitor collects similar information at a summarized level in memory and then dumps the data into multiple tables
    - Interface: Operations Navigator & APIs

Query Performance Tuner - QAQQINI

- Provides central point of control for all attributes, options, and knobs that can impact query opitmization
  - Table design allows attributes to be set dynamically with just database updates or insert/delete

```
     UPDATE mylib/QAQQINI SET QQVAL='600'
          WHERE QQPARM='QUERY_TIME_LIMIT';

    INSERT mylib/QAQQINI
         VALUES('MESSAGES_DEBUG','*YES','Activated - 4pm');
```

  - One row per attribute/parm and 3 character columns
    - QQPARM - the attribute/option name
    - QQVAL - value of the attribute/option
    - QQTEXT - optional description of the attribute or it values

QQPARM	QQVAL	QQTEXT
MESSAGES_DEBUG	*YES	Debug Set - 11pm
QUERY_TIME_LIMIT	600	New time limit - set 7/25
PARALLEL_DEGREE	*DEFAULT	
FORCE_JOIN_ORDER	*DEFAULT	
...	...	

Visual Explain

- Visualization of the query access plan
  - Details and attributes of the query plan, execution, and database objects involved
  - V5R1 includes auto-highlighting of icons
- Visual Explain can be used in one of two ways
  - Interactively with Ops Navigator SQL Script window
  - Reactively based on previously collectly database monitor data (detailed monitor)
- Requires V4R5 or higher of OS/400 and IBM iSeries Navigator

Tuning Tools Comparison Table

PRTSQLINF	STRDBG/CHGQRYA QAQQINI	STRDBMON	Memory -based Monitor
Available without running query (after access plan has been created)	Only available when the query is run	Only available when the query is run	Only available when the query is run
Displayed for all queries in SQL pgm or pkg, whether executed or not	Displayed only for those queries which are executed	Displayed only for those queries which are executed	Displayed only for those queries which are executed
Information on host variable implementation	Limited information on the implementation of host variables	All information on host variables, implementation, and values	All information on host variables, implementation and values
Available only to SQL users with pgms, packages, or service pgms	Available to all query users (OPNQRYF, SQL, QUERY/400)	Available to all query users (OPNQRYF, SQL, QUERY/400)	Available only to SQL interfaces
Messages printed to spool file	Messages displayed in job log	Performance records written to database file	Performance information collected in memory and then written to database file
Easier to tie messages to query with subqueries or unions	Difficult to tie messages to query with subqueries or unions	Uniquely identifies every query	Repeated query requests are summarized

Additional Resources

- DB2 UDB for iSeries home page: www.iseries.ibm.com/db2
- iSeries SQL Performance Workshop (Course #S6140)
 http://www-1.ibm.com/servers/eserver/iseries/service/igs/db2performance.html
- Online DB2 UDB publications www.iseries.ibm.com/db2/books.htm
  - Database Performance & Query Optimization
- SQL Interface FAQs:
  - CLI : www.iseries.ibm.com/db2/clifaq.htm
  - JDBC
    - Toolbox: www.iseries.ibm.com/toolbox/faqjdbc.htm
    - Native: www.iseries.ibm.com/developer/jdbc/index.html
- QAQQINI script builder:
  www.iseries.ibm.com/developer/bi/tuner.html
- DB2 UDB for iSeries Online Education
  www.iseries.ibm.com/developer/education/ibo/view.html?biz
- Third-pary performance tools:
  - Centerfield Technology (www.centerfieldtechnology.com)