Toronto Users Group
March 19, 2014
The Science and Art of Indexing on DB2 for i

Speaker Name Linda M Swan
lmswan@us.ibm.com

Scenario

Find the first occurrence of “**IBM**” in a very large book…

What do you do first?

Turn to the index!

in·dex Something that serves to guide, point out, or otherwise facilitate efficient reference.

Creating a useful index

is both a *Science* and an *Art*.

Agenda

- DB2 for i Indexing Technology
- Query Optimization using Indexes
- Indexing Strategies
- Case Study

Indexing Technology
within DB2 for i

DB2 for i

- Two types of indexing technologies are supported
  - ***Radix*** Index
  - ***Encoded Vector*** Index

  (OmniFind Text Search Server. See reference page)

- Each type of index has specific uses and advantages
- Respective indexing technologies compliment each other
- Indexes can be used for statistics and implementation
- Indexes can provide RRNs and/or data
- Indexes are scanned or probed
  - Probe can only occur on contiguous, leading key columns
  - Scan can occur on any key column
  - Probe and scan can be used together

Using Indexes - Probe v Scan

- **Probe** (key positioning)
with leading, n contiguous
key columns
1
1+2
1+2+3

- **Scan** (test)
with any other
key columns
2
3
2+3

Index Key Columns (ITEM_NO, COLOR, SIZE)

ITEM_NO	COLOR	SIZE
001	BLUE	LARGE
002	RED	SMALL
003	BLACK	SMALL
004	GREEN	MEDIUM

...WHERE COLOR = 'BLACK‘ AND ITEM_NO = 003

...WHERE SIZE = 'MEDIUM'

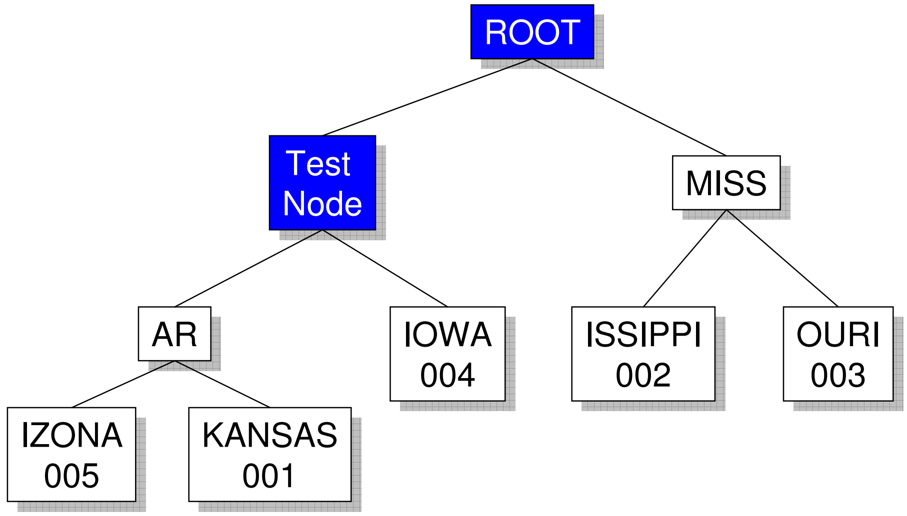
...WHERE ITEM_NO = 001 AND SIZE = ‘LARGE’

Radix Index

- Index “tree” structure
- Key values are compressed
  - Common patterns are stored once
  - Unique portion stored in “leaf” pages
  - Positive impact on size and depth of the index tree
- Algorithm used to find values
  - Binary search
  - Modified to fit the data structure
- Maintenance
  - Index data is automatically spread across all available disk units
  - Tree is automatically rebalanced to maintain an efficient structure
- Temporary indexes
  - Considered a temporary data structure to assist the DB engine
  - Maintained temporary indexes available in SQE
  - Goes away at IPL and at the discretion of the optimizer

Radix Index

Database Table	
001	ARKANSAS
002	MISSISSIPPI
003	MISSOURI
004	IOWA
005	ARIZONA
…	…



ADVANTAGES:
- Very fast access to a **single** key value
- Also fast for **small, selected range** of key values (low cardinality)
- Provides order

DISADVANTAGES:
- Table rows retrieved in order of key values (not physical order) which equates to **random I/O's**
- No way to predict which physical index pages are next when traversing the index for large number of key values
  - Optimizer will add RIO nodes

Index Probe Example

Given an index on table EMPLOYEE keyed on STATE…

SELECT *
FROM EMPLOYEE
WHERE STATE = 'IOWA'

EMPLOYEE Index

STATE
…
IOWA (004)
IOWA (017)
IOWA (007)
IOWA (010)
KANSAS (011)
MISSISSIPPI (002)
MISSISSIPPI (013)
MISSOURI (003)
…

RRN

EMPLOYEE Table

RRN	STATE
001	ARKANSAS
002	MISSISSIPPI
003	MISSOURI
004	IOWA
005	ARIZONA
006	MONTANA
007	IOWA
008	NEBRASKA
009	NEBRASKA
010	IOWA
011	KANSAS
012	WISCONSIN
013	MISSISSIPPI
014	WISCONSIN
015	WISCONSIN
016	ARKANSAS
017	IOWA

Perform a probe into the range using the local selection value(s)

Encoded Vector Index (EVI)

- Index for delivering fast data access in analytical and reporting environments
  - Advanced technology from IBM Research
  - Used to produce dynamic bitmaps and RRN lists
  - Fast access to statistics to improve query optimizer decision making
- Not a “tree” structure
- Can only be created through an SQL interface or Navigator for i GUI

```
CREATE ENCODED VECTOR INDEX MySchema.IXName
                    ON MySchema.TabName(KEY(s))

                    INCLUDE ( SUM(SomeOtherColName));
```

New in 7.1
Maintained aggregate

Encoded Vector Index (EVI)

Symbol Table				
Key Value	Code	Count	Include Sum()	Include Sum()
Arizona	1	5000	1500	2005
Arkansas	2	7300	3200	450
…				
Wisconsin	49	340	575	1200
Wyoming	50	2760	210	0

optional

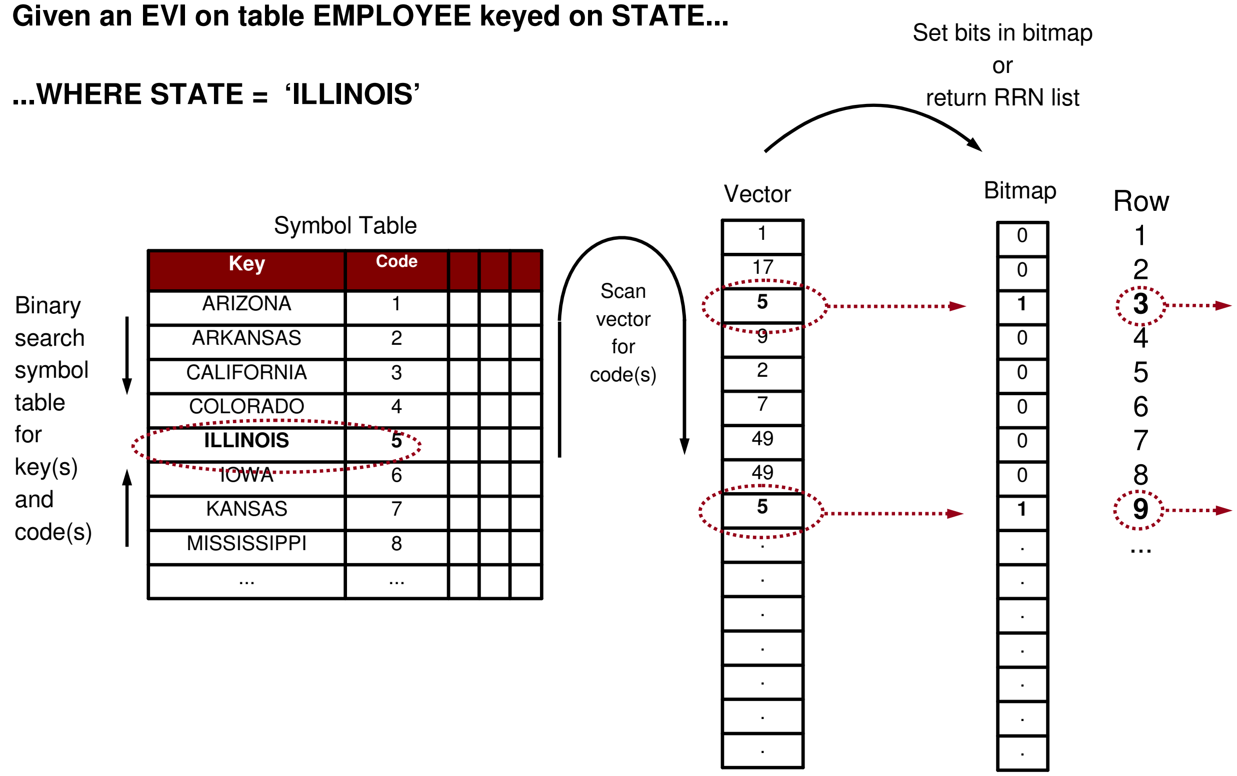
Vector	RRN
1	1
17	2
5	3
9	4
2	5
7	6
50	7
49	8
5	9
…	…

- Symbol table contains information for each distinct key value
  - Each key value is assigned a unique 1,2, or 4 byte code (key compression)
  - Enhanced in i 7.1 to INCLUDE SUM and COUNT in the definition
- Rather then a bit array for each distinct key value, use one array of codes

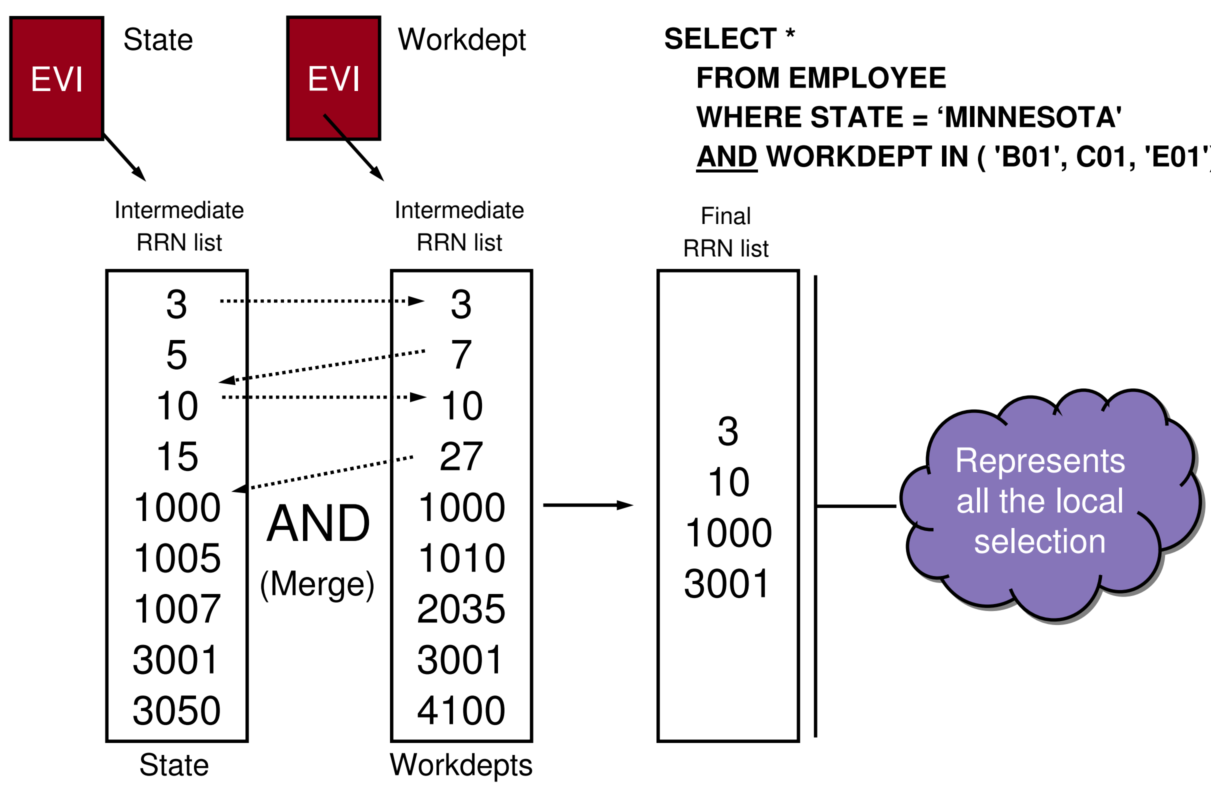
Bitmap / RRN List Example

Given an EVI on table EMPLOYEE keyed on STATE...

...WHERE STATE = 'ILLINOIS'

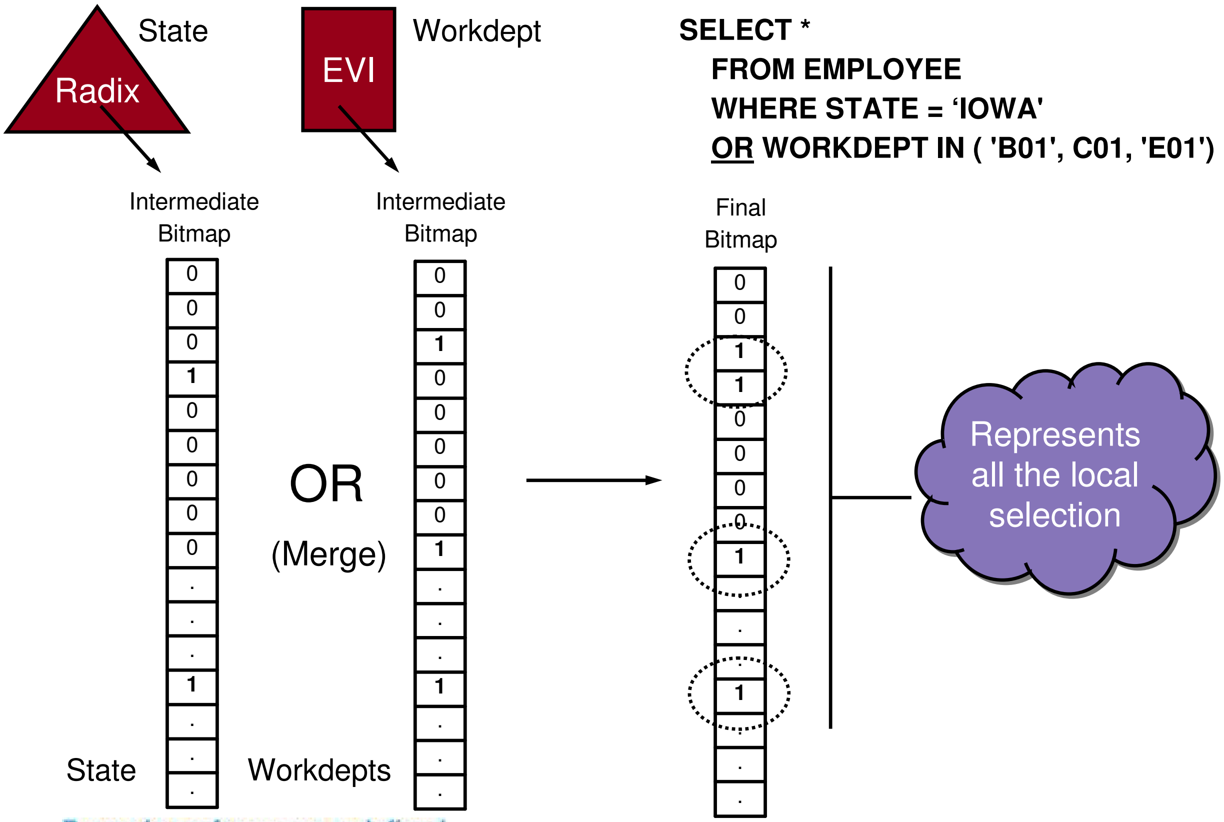


Index ANDing Example



Index ORing Example

```
SELECT *
  FROM EMPLOYEE
  WHERE STATE = 'IOWA'
  OR WORKDEPT IN ( 'B01', C01, 'E01')
```



EVI Symbol Table Only Example

Given an EVI on table EMPLOYEE keyed on STATE…

SELECT COUNT(*) FROM EMPLOYEE WHERE STATE = 'Wisconsin';

SELECT COUNT(DISTINCT STATE) FROM EMPLOYEE;

SELECT STATE, SUM(commission), SUM(salary) FROM EMPLOYEE GROUP BY STATE;

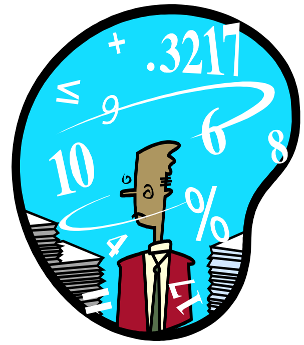
Search / Scan symbol table for key(s) and counts

Symbol Table				
Key Value	Code	Count	Include Sum()	Include Sum()
Arizona	1	5000	1500	2005
Arkansas	2	7300	3200	450
…				
Wisconsin	49	340	575	1200
Wyoming	50	2760	210	0

DB2 for IBM i

cardinality The number of distinct elements in a set.

- High cardinality = large distinct number of values
- Low cardinality = small distinct number of values



In general…

- A radix index is best when accessing a small set of rows and the key cardinality is high
- An encoded vector index is best when accessing a set of rows and the key cardinality is low
- Understanding the data and query are key

Creating Indexes

- CREATE INDEX SQL statement

  CREATE INDEX MY_IX on MY_TABLE (KEY1, KEY2)

- CREATE ENCODED VECTOR INDEX SQL statement

  CREATE ENCODED VECTOR INDEX MY_EVI on MY_TABLE (KEY1)

- IBM i Navigator – client based database graphical interface
- IBM Navigator for i – browser based
- CRTPF and CRTLF CL commands
  - Keyed access path within the physical file or logical file
  - Join logical file
- Primary Key, Foreign Key and Unique Key Constraints
  - CREATE TABLE
  - ALTER TABLE
  - ADDPFCST

6.1 Creation of Index with Derived Keys

- Creation of indexes with *derived* keys via SQL

CREATE INDEX ORDERPRIORITYUPPER ON T1
(**UPPER(ORDERPRIORITY)** AS UORDERPRIORITY ASC);

CREATE ENCODED VECTOR INDEX YEARQTR ON T1
(**YEAR(ORDERDATE)** AS ORDYEAR ASC,
QUARTER(ORDERDATE) AS ORDQTR ASC);
- timestamp field an even better example here

CREATE INDEX TOTALEXTENDEDPRICE ON T1
(**QUANTITY * EXTENDEDPRICE** AS TOTEXTPRICE ASC);

NOTE: There are some restrictions on when indexes can be matched by the optimizer to the query in 6.1

6.1 Create **Sparse** Indexes from SQL

- Support of WHERE clause on SQL create index

CREATE INDEX FASTDELIVER ON T1
(SHIPMODE ASC)
WHERE SHIPMODE = 'NEXTDAYAIR'
OR SHIPMODE = 'COURIER';

NOTE: Sparse Indexes are *NOT* used by the query optimizer prior to 7.1

NOTE: DB2 for i Optimizer team recommends a good general purpose indexing strategy over reliance on the use of sparse indexes

When to a use Derived Index?

- Could replace some logical files with SQL indexes for use by RLA native, high level language programs
  - Modernize those objects
  - Big logical page size (8K v **64K**)
    - A keyed LF will share the access path of an SQL created index, but reverse is not true

- Derived indexes may be useful for
  - Case insensitive searches
  - Data extracted from a column (i.e. SUBSTR, YEAR, MONTH…)
  - Derive Common Grouping columns (i.e. YEAR(ORDERDATE))
  - Results of operations (COL1+COL2 , QTY * COST)
  - Might be useful to allow *index only access* in more cases
    - Especially with INCLUDE support FOR 7.1
  - **Reduce table scans, index scans and temporary data structures**

EVI's and Grouping

- EVI with A, B, C key fields and INCLUDE(SUM(D)…)
  Create encoded vector index GBEVI02 on T1 (A ,B , C) **INCLUDE(SUM(D))**

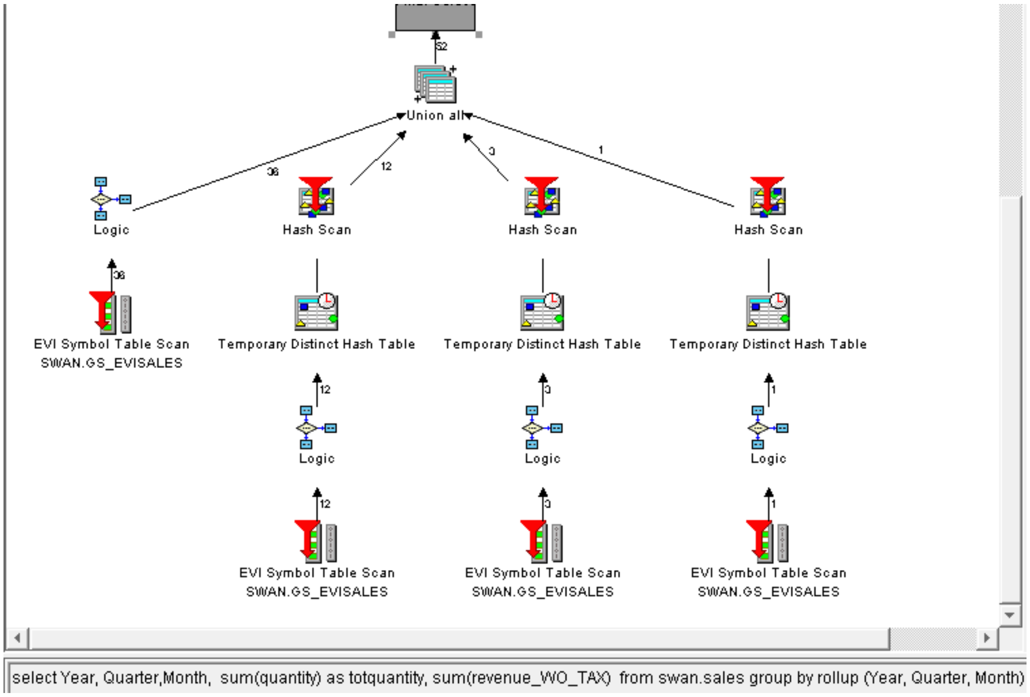
Will be usable for group by ALL Grouping combinations of A,B,C (including Grouping set combinations)

Example:

```
SELECT A,B,C, SUM(D) FROM T1 GROUP BY GROUPING SETS((A), (B), (C))
SELECT A,B,C, SUM(D) FROM T1 GROUP BY GROUPING SETS((A), (B))
SELECT A,B,C, SUM(D) FROM T1 GROUP BY GROUPING SETS((A), (C))
SELECT A,B,C, SUM(D) FROM T1 GROUP BY GROUPING SETS((B), (C))
SELECT A,B,C, SUM(D) FROM T1 GROUP BY GROUPING SETS((A,B), (C))
SELECT A,B,C, SUM(D) FROM T1 GROUP BY GROUPING SETS((A,C), (B))
SELECT A,B,C, SUM(D) FROM T1 GROUP BY ROLLUP(A,B,C)
SELECT A,B,C, SUM(D) FROM T1 GROUP BY CUBE(A,B,C)

SELECT A,B,C, SUM(D) FROM T1 GROUP BY (A,B,C)
SELECT A,B,    SUM(D)  FROM T1 GROUP BY (A,B)
SELECT C       SUM(D)  FROM T1 GROUP BY (C)    /* Or A, Or B */
SELECT         SUM(D) FROM T1
```

- Experiment with the NEW EVI INCLUDE support on DB2 for IBM i for Grouping and Grouping SET Queries
  - Create EVI with common GB columns and INCLUDE most commonly used sums
  - Always add in COUNT(*) to EVI INCLUDE
  - **Do this only for GB columns that have relatively small cardinality**
  - **EVIs work best if NOT constantly adding new Key values**



CREATE ENCODED VECTOR INDEX swan.GS_EVISales ON .sales (YEAR ASC, QUARTER ASC, MONTH ASC) INCLUDE (SUM(QUANTITY) , sum(REVENUE_WO_TAX), count(*)) ;

IBM Power Systems

7.1 Index Enhancements

Other Index related enhancements

- **In-Memory Table/Index (7.1)**

  CHGPF FILE(MYSCHEMA/TAB1) KEEPINMEM(*YES)
  CHGLF FILE(MYSCHEMA/IX1) KEEPINMEM(*YES)

- **SSD Support for Table/Index (6.1)**

  CHGLF FILE(MYSCHEMA/IX1) UNIT(*SSD)

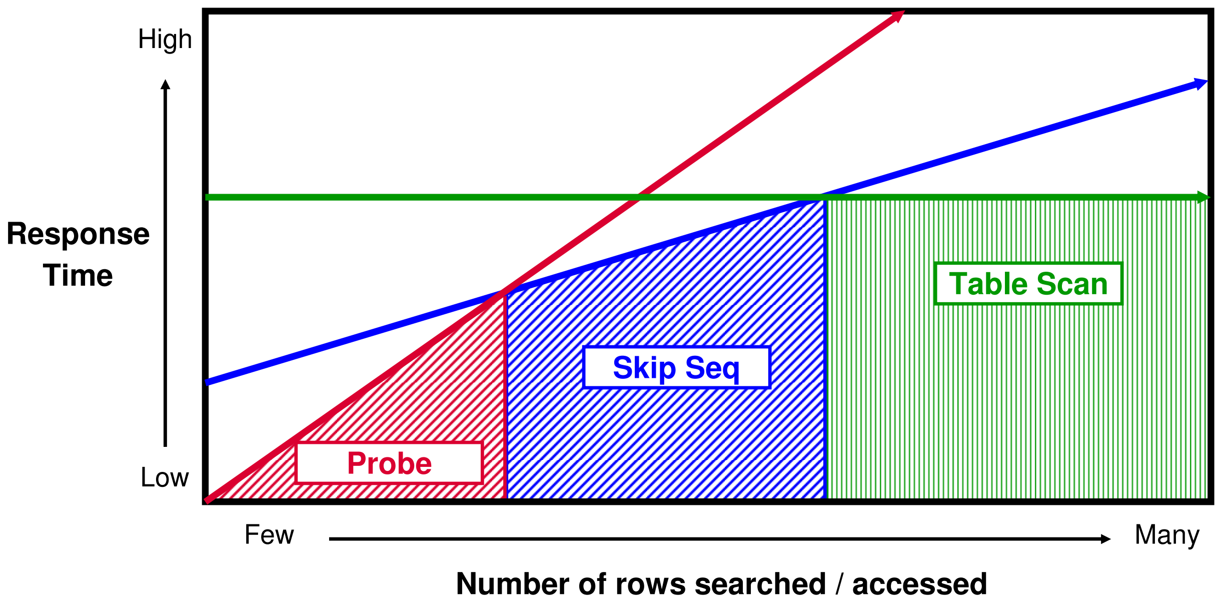
- **Expanded Optimizer matching of Sparse and derived Indexes (7.1)**

  CREATE INDEX cust_act ON CUSTIMERS(cust_id) WHERE activCust='Y'

Query Optimization (using indexes)

Data Access Methods

Cost based optimization dictates that the fastest access method for a given table will vary based upon *selectivity* of the query



Strategy for Query Optimization

Query optimization will generally follow this simplified strategy:

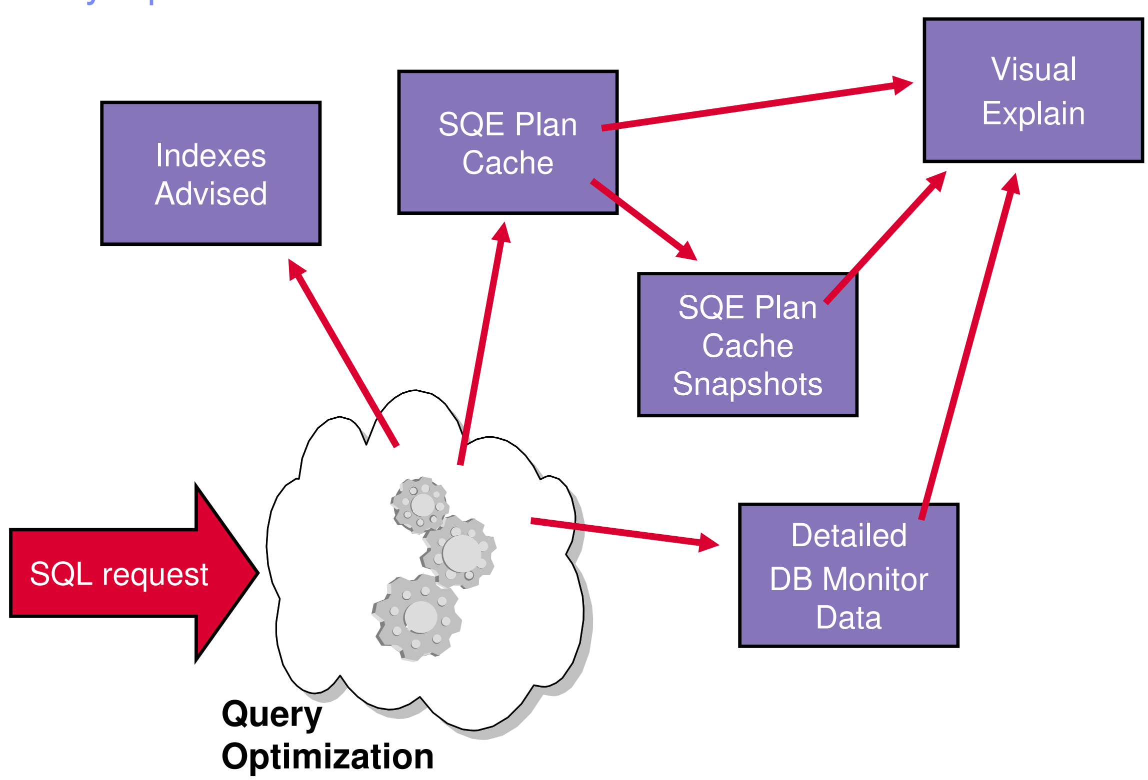
- ✓Gather meta-data and statistics for costing
  - Selectivity statistics
  - Indexes available for implementation to be costed
    - Sort the indexes based upon their usefulness
    - Remove indexes that 'cover' other indexes
  - Environmental attributes that may affect the costs
- ✓Generate default cost
  - Build an access plan associated with the default plan
- ✓For each index:
  - Gather information needed specific to this index
  - Build an access plan based on this index
  - Cost the use of the index with this access plan
  - Compare the resulting cost against the cost from the current best plan

Strategy for Query Optimization

Optimizing indexes will generally follow this simplified strategy:

- Gather list of indexes for statistics and costing
- Sort the list of indexes considering how the index can be used
  - Local selection
  - Joining
  - Grouping
  - Ordering
  - Index only access
- One index may be useful for statistics, and another useful for implementation

Query Optimization Feedback

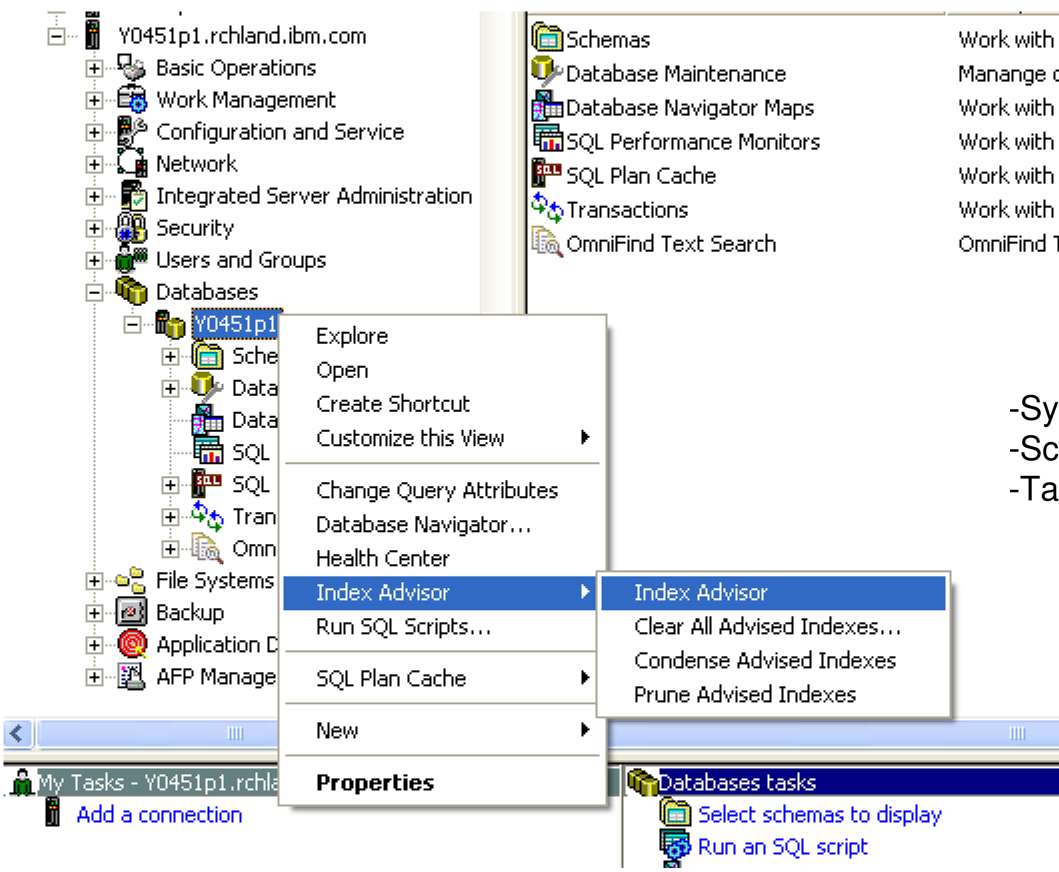


Indexing Advice from the Optimizer

- SQE provides index creation advice
  - QSYS2/SYSIXADV – system wide, always on
  - i Navigator –via visual explain

- SQE
  - Robust advice
  - Radix and EVI indexes
  - Based on all parts of the query
    - Local selection
    - Join
    - Grouping
    - Ordering
    - Includes OR predicate advice
  - Multiple indexes can be advised for the same query
  - Some limitations
    - Optimizer doesn't advise EVI with INCLUDE clause
    - Optimizer doesn't advise derived or sparse
    - Optimizer doesn't advise specifically for Index Only Access

Index Advised – System wide



-System
-Schema
-Table level

Index Advised – System wide

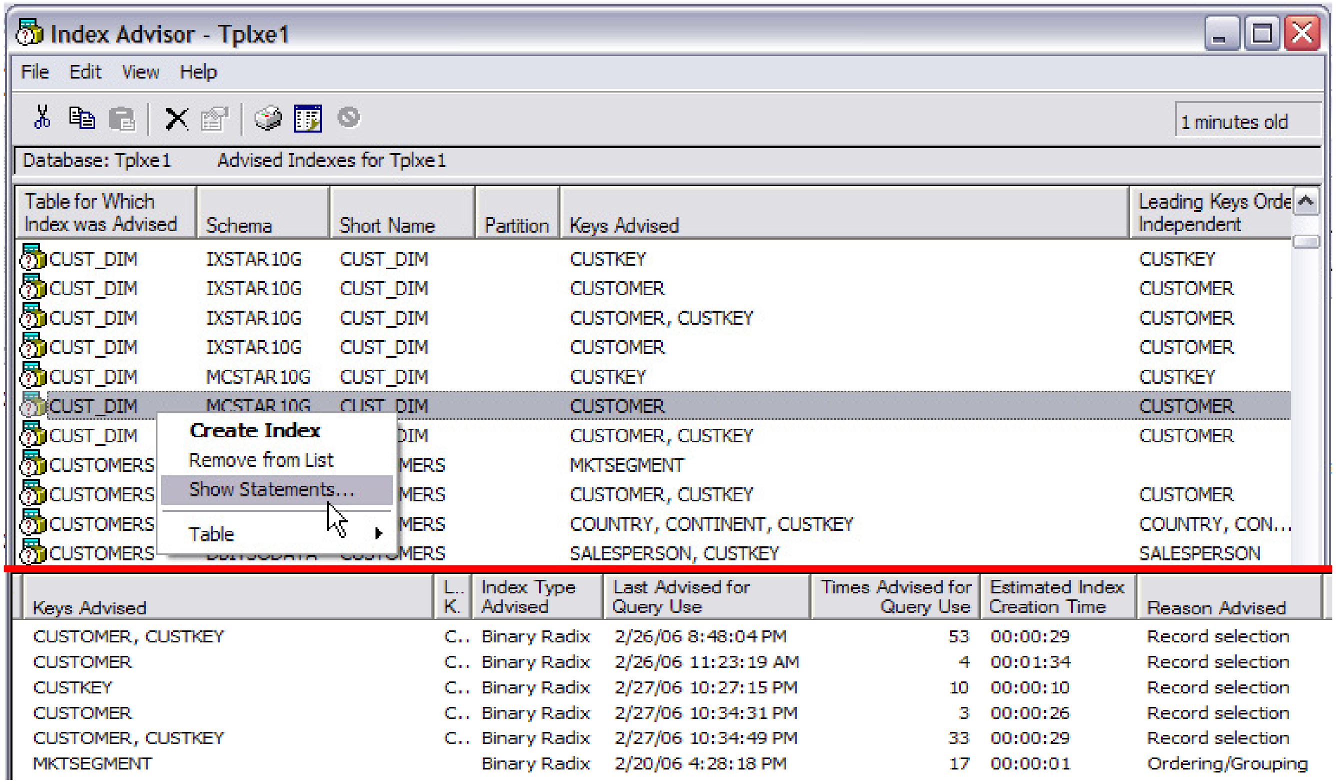


Table for Which Index was Advised	Schema	Short Name	Partition	Keys Advised	Leading Keys Order Independent
CUST_DIM	IXSTAR10G	CUST_DIM		CUSTKEY	CUSTKEY
CUST_DIM	IXSTAR10G	CUST_DIM		CUSTOMER	CUSTOMER
CUST_DIM	IXSTAR10G	CUST_DIM		CUSTOMER, CUSTKEY	CUSTOMER
CUST_DIM	IXSTAR10G	CUST_DIM		CUSTOMER	CUSTOMER
CUST_DIM	MCSTAR10G	CUST_DIM		CUSTKEY	CUSTKEY
CUST_DIM	MCSTAR10G	CUST_DIM		CUSTOMER	CUSTOMER
CUST_DIM		DIM		CUSTOMER, CUSTKEY	CUSTOMER
CUSTOMERS		MERS		MKTSEGMENT	
CUSTOMERS		MERS		CUSTOMER, CUSTKEY	CUSTOMER
CUSTOMERS		MERS		COUNTRY, CONTINENT, CUSTKEY	COUNTRY, CON...
CUSTOMERS		MERS		SALESPERSON, CUSTKEY	SALESPERSON

Keys Advised	L.. K.	Index Type Advised	Last Advised for Query Use	Times Advised for Query Use	Estimated Index Creation Time	Reason Advised
CUSTOMER, CUSTKEY	C..	Binary Radix	2/26/06 8:48:04 PM	53	00:00:29	Record selection
CUSTOMER	C..	Binary Radix	2/26/06 11:23:19 AM	4	00:01:34	Record selection
CUSTKEY	C..	Binary Radix	2/27/06 10:27:15 PM	10	00:00:10	Record selection
CUSTOMER	C..	Binary Radix	2/27/06 10:34:31 PM	3	00:00:26	Record selection
CUSTOMER, CUSTKEY	C..	Binary Radix	2/27/06 10:34:49 PM	33	00:00:29	Record selection
MKTSEGMENT		Binary Radix	2/20/06 4:28:18 PM	17	00:00:01	Ordering/Grouping

Index Advice

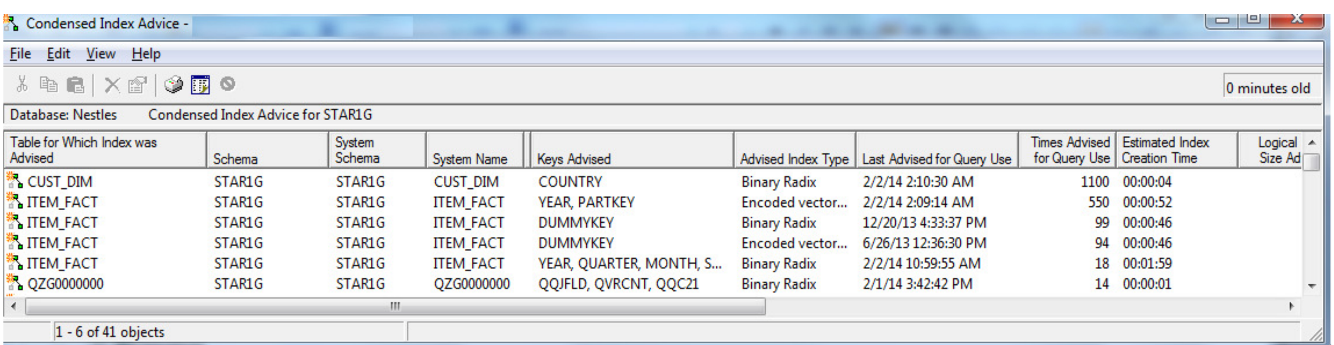


Table for Which Index was Advised	Schema	System Schema	System Name	Keys Advised	Advised Index Type	Last Advised for Query Use	Times Advised for Query Use	Estimated Index Creation Time	Logical Size Ad
CUST_DIM	STAR1G	STAR1G	CUST_DIM	COUNTRY	Binary Radix	2/2/14 2:10:30 AM	1100	00:00:04	
ITEM_FACT	STAR1G	STAR1G	ITEM_FACT	YEAR, PARTKEY	Encoded vector...	2/2/14 2:09:14 AM	550	00:00:52	
ITEM_FACT	STAR1G	STAR1G	ITEM_FACT	DUMMYKEY	Binary Radix	12/20/13 4:33:37 PM	99	00:00:46	
ITEM_FACT	STAR1G	STAR1G	ITEM_FACT	DUMMYKEY	Encoded vector...	6/26/13 12:36:30 PM	94	00:00:46	
ITEM_FACT	STAR1G	STAR1G	ITEM_FACT	YEAR, QUARTER, MONTH, S...	Binary Radix	2/2/14 10:59:55 AM	18	00:01:59	
QZG0000000	STAR1G	STAR1G	QZG0000000	QQJFLD, QVRCNT, QQC21	Binary Radix	2/1/14 3:42:42 PM	14	00:00:01	

MTI Used	MTI Created	MTI Last Used
131072	1	11/30/13 11:50:07 AM
56215	1	10/21/13 10:23:09 AM
50870	1	10/10/13 4:39:29 PM
0	0	1/1/01 12:00:00 AM
45751	1	9/10/13 1:49:01 PM
0	0	1/1/01 12:00:00 AM
8	2	5/24/13 1:54:56 PM

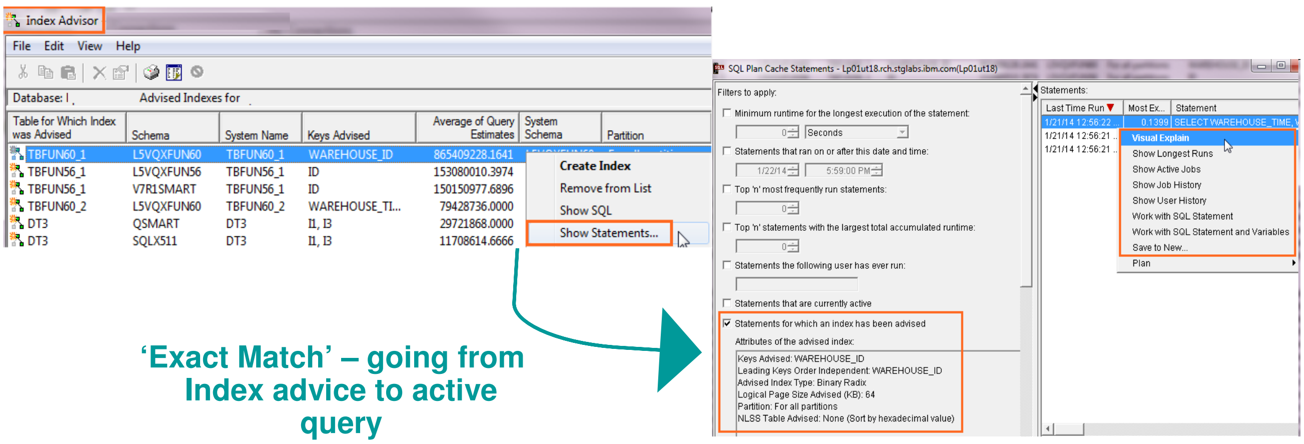
If an advised index is created many times as a MTI and/or used often, consider making it permanent

If an Index is advised a large number of time.

Index Advisor → Show Statements - improved query identification

Launch into Show Statements from the Index Advisor

- Show Statements will find queries per the LIVE plan cache based upon how its launched:
  1. Launch Show Statements directly (**table match**)
  2. Launch from Index Advice (**exact match**)
  3. Launch from Condensed Index Advice (**fuzzy match**)
     - Queries which match any ordering or subset of the keys



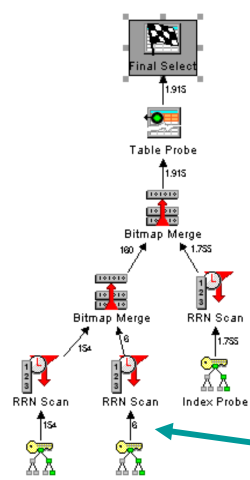
'Exact Match' – going from Index advice to active query

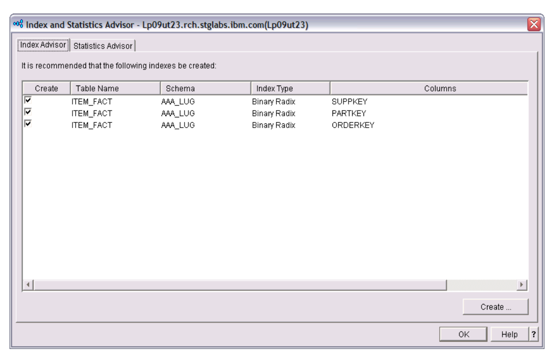
Improved index advice generation to handle OR predicates

Index OR Advice example

- Should advise indexes over all 3 OR'ed predicate columns
- All 3 advised indexes will have **DEPENDENT_ADVICE_COUNT > 0**
- Execution with indexes should produce bitmap implementation and register no new advice

select orderkey, partkey, suppke
 linenumber, shipmode order
from ABC_ITEM_fact
where **OrderKey <= 10 OR**
 SuppKey <= 10 OR
 PartKey <= 10
optimize for all rows





Times Advised for Query Use	Times Advised Dependent on	Average of Query Estimates
1307	1163	400.7699

Visual Explain

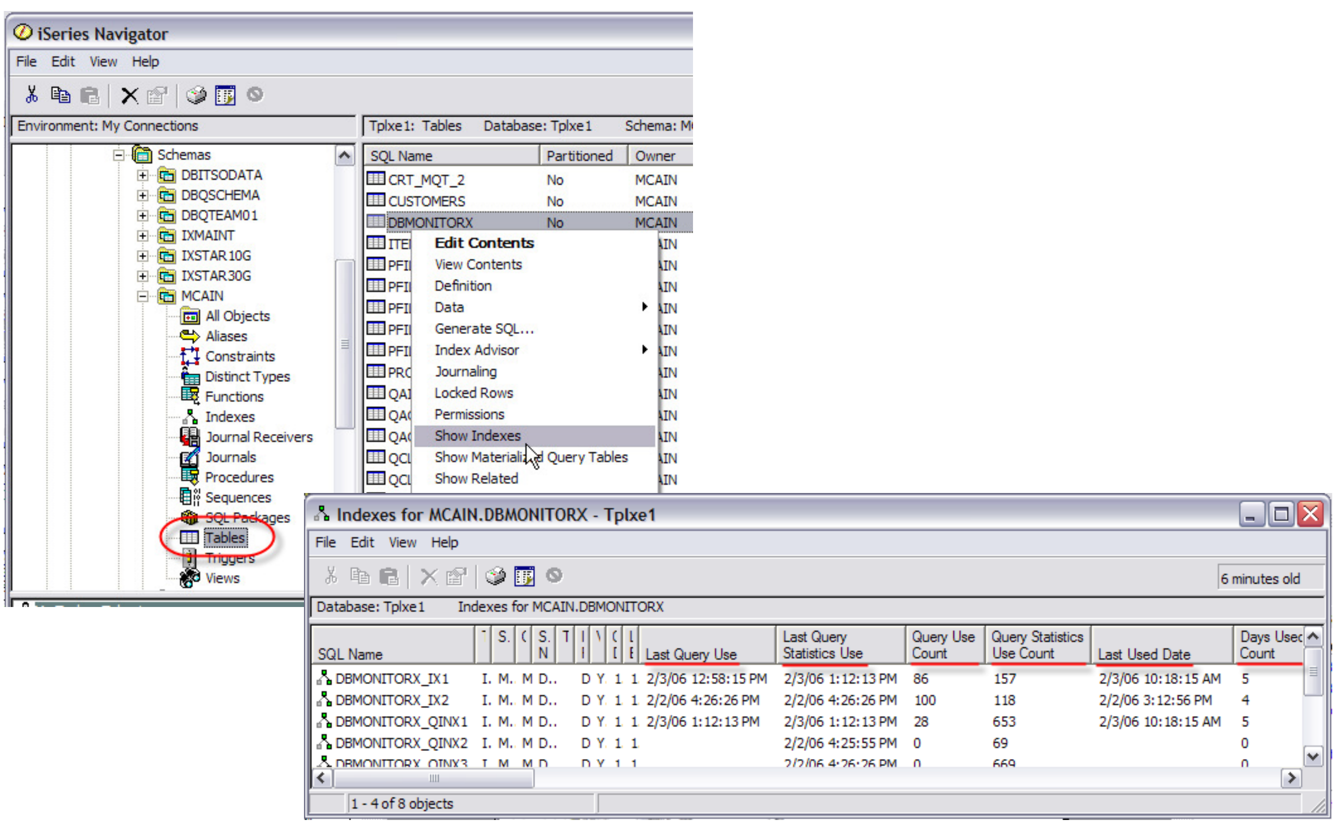
Implementation … Advice

Index Advisor indicator of OR advice

Maintained Temporary Indexes (MTIs)

- Optimizer can request the DB Engine create a temporary index
- Both full and sparse indexes can be created
  SQe only create sparse for ordering and for queries running with live data mode, no QDS (temps) allowed, sparse MTIs are not reusable
- SQE Temporary indexes (MTIs) are also used for statistics in i7.1
- Temporary indexes are *maintained*
- SQE
  - Temporary indexes are reused and shared across jobs and queries
  - Creation is based on "watching" the query requests over time
  - Creation is based on optimizer's own index advice
  - Temporary index maintenance is delayed when all associated cursors closed

Index Evaluator – Show Indexes tells us when an index is not used



Index Evaluator via Catalog Views

SYSINDEXSTAT	Contains one row for every SQL index. Use this view when you want to see information for a specific SQL index or set of SQL indexes. The information is similar to that returned via Show Indexes in IBM i Navigator.
SYSPARTITIONINDEXES	Contains one row for every index built over a table partition or table member. Use this view when you want to see index information for indexes built on a specified table or set of tables. The information is similar to that returned via Show Indexes in IBM i Navigator.
SYSTABLEINDEXSTAT	Contains one row for every index that has at least one partition or member built over a table. If the index is over more than one partition or member, the statistics include all those partitions and members.

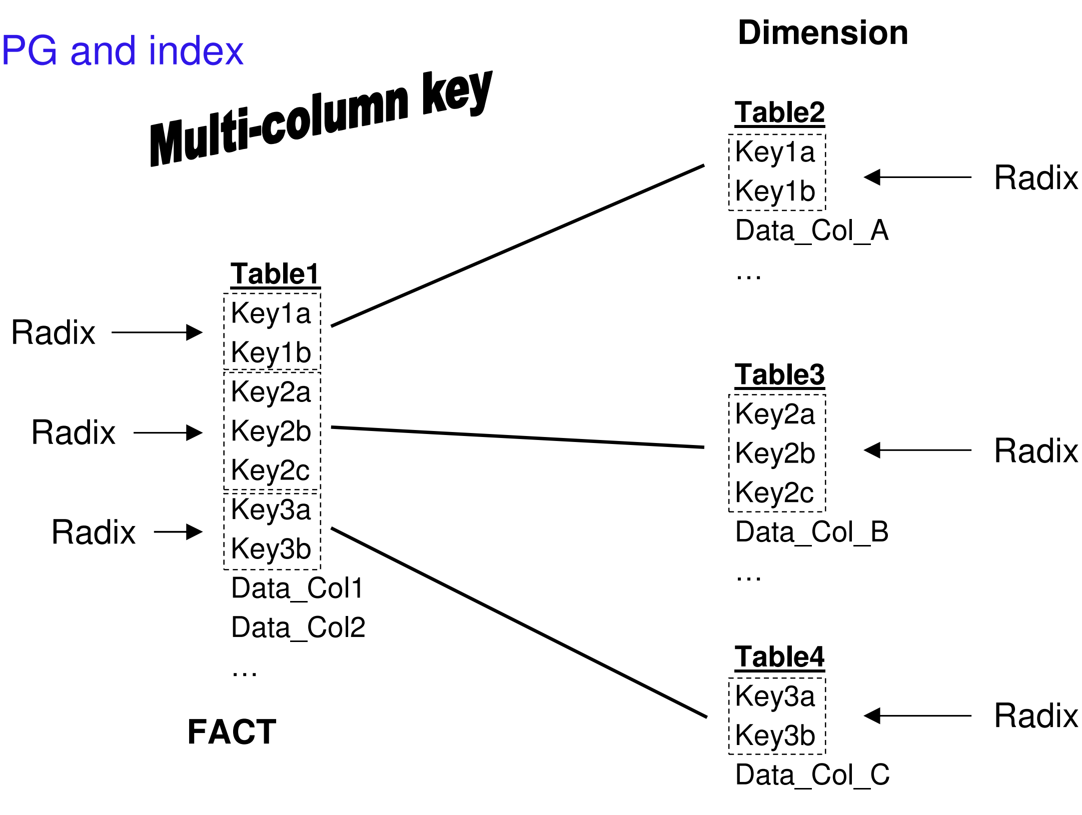
Lookahead Predicate Generation Technology

Look-ahead Predicate Generation (LPG)

- **A strategy to *generate* local selection predicates for one table, from one or more other tables**
- **Using the (generated) local selection predicates, more options are available for data access and data processing**
- **Minimizes the effects of a suboptimal join order**
- **Opportunity for additional indexing**
- **Can a have a very positive affect on query performance!**
- **LPG is an example of query rewrite technique unique to DB2 for i**

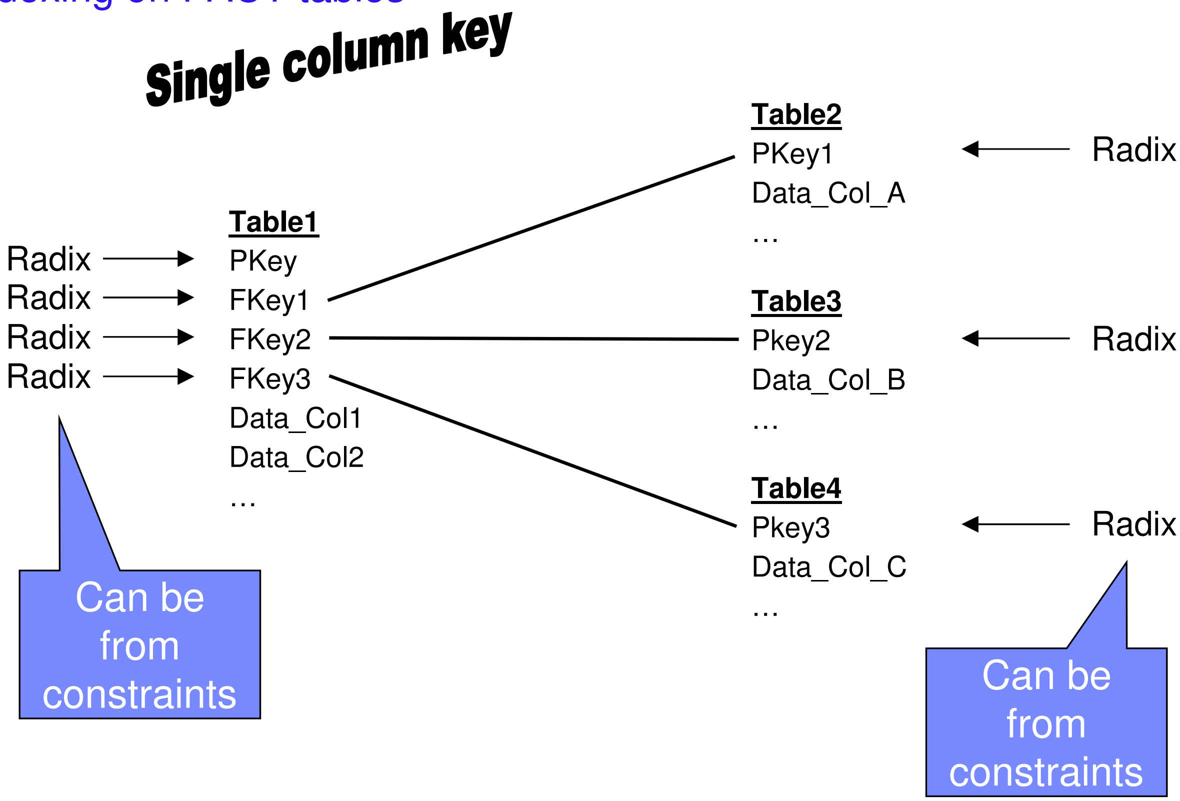
- **Only available for single column join conditions!**

LPG and index

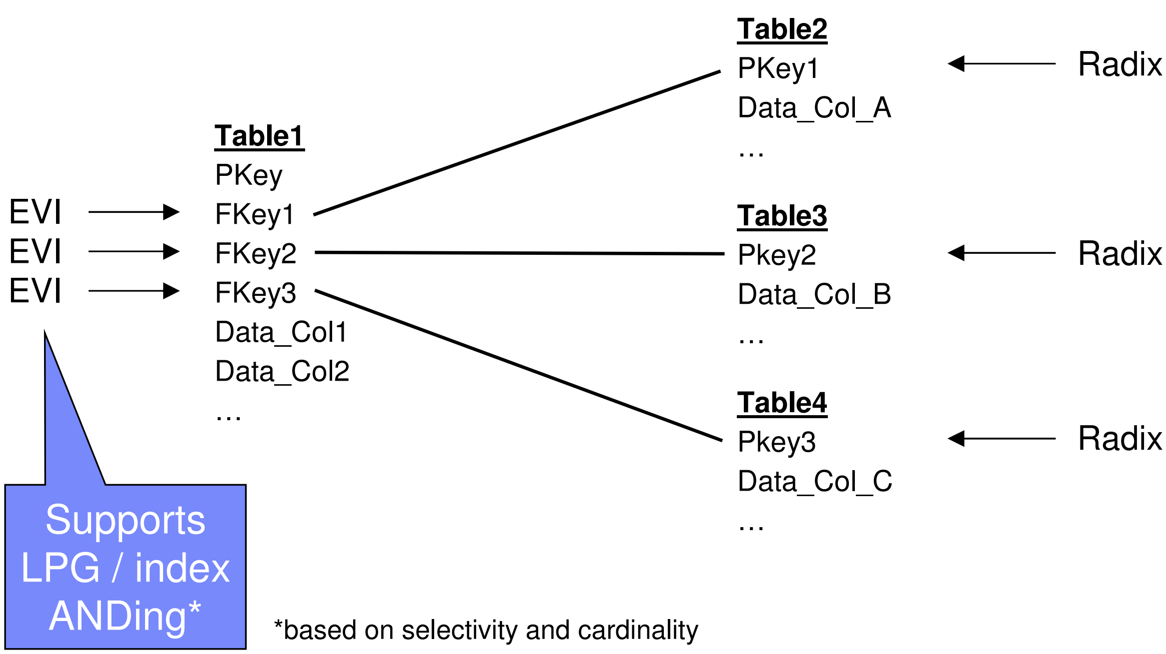


If the relationship between the Fact and the Dimension tables is multi-key LPG will not be applied

Indexing on FACT tables



Indexing on FACT tables



*based on selectivity and cardinality

Indexing Strategies

What should you do?

Create all advised indexes?

Nothing, let the system handle it?

Monitor, analyze, and tune important tables and queries?

DB2 for i

The goals of creating indexes are:

1. Provide the optimizer the **statistics** needed to understand the data, based on the query
2. Provide the optimizer **implementation** choices, based on the selectivity of the query

✓Accurate statistics means accurate costing

✓Accurate costing means optimal query plan

✓Optimal query plans means happy customer

The Process of Identifying Indexes

Proactive method
- Analyze the data model, application and SQL requests

Reactive method
- Rely on optimizer feedback and actual implementation methods
- Rely on SQE's ability to auto tune using temporary indexes

Understand the data being queried
- Column selectivity
- Column cardinality

Separating complex queries into individual parts by table
- Selecting
- Joining
- Grouping
- Ordering
- Subquery
- View

Indexing Strategy - Basic Approach

Radix Indexes
- Common local selection columns
- Join columns

} Minimum

- Local selection columns + join columns
- Local selection columns + grouping columns
- Local selection columns + ordering columns

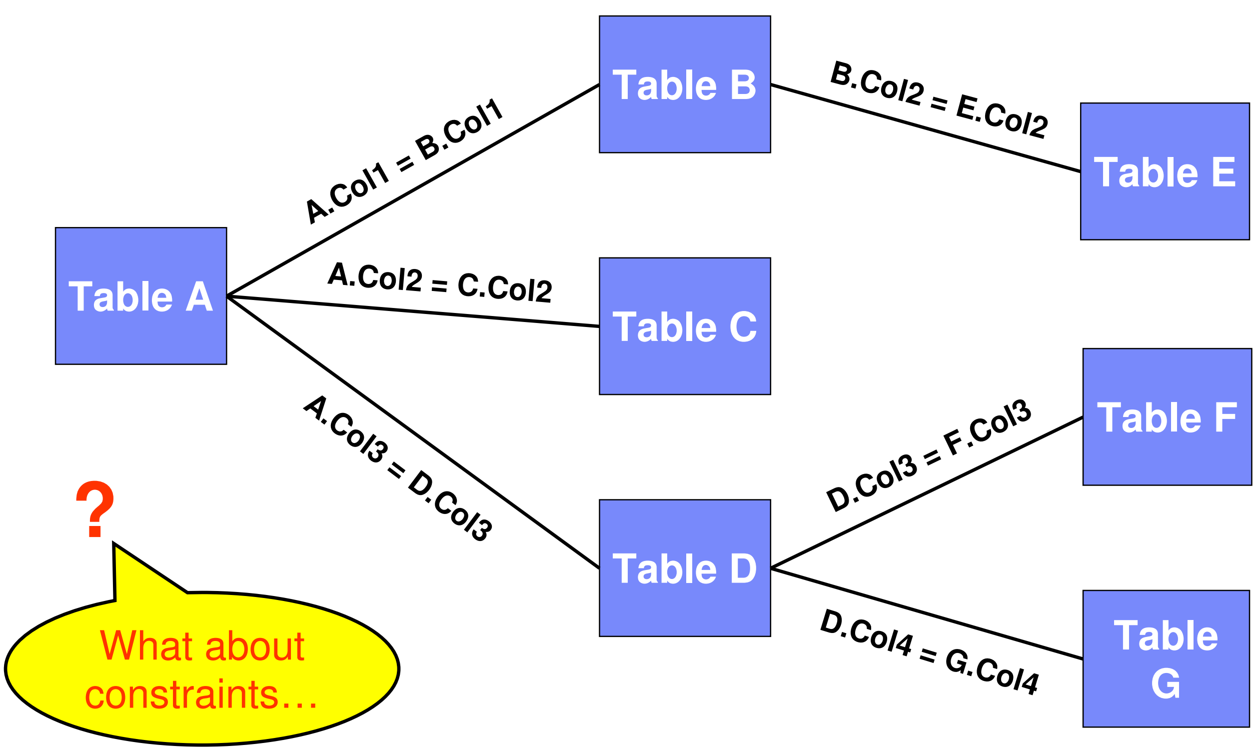
Advanced

Requires knowledge of query optimization and data lifecycle

Encoded Vector Indexes
- Local selection column for index ANDing/ORing
- Join columns (star or snowflake schema)
- Index only access
  - DISTINCT, COUNT, COUNT DISTINCT, SUM()

Note: Columns used with equal conditions are first in key list

Indexing Strategy - Examples



Indexing Strategy - Examples

```
-- Query 1
SELECT      A.CUSTOMER_NO, A.ORDER_DATE, A.QUANTITY
FROM        ORDERS A
WHERE       A.CUSTOMER_NO = 0112358;

CREATE INDEX ORDERS_IX1 ON ORDERS (CUSTOMER_NO);


-- Query 2
SELECT      A.CUSTOMER_NO, A.ORDER_DATE, A.QUANTITY
FROM        ORDERS A
WHERE       A.CUSTOMER_NO = 0112358
AND         A.ITEM_ID = 'ABC123YXZ';

CREATE INDEX ORDERS_IX2 ON ORDERS (CUSTOMER_NO, ITEM_ID);
```


Indexing Strategy - Examples

```
-- Query 3
SELECT      A.CUSTOMER_NO, A.CUSTOMER, A.ORDER_DATE
FROM        ORDERS A
WHERE       A.CUSTOMER_NO IN (0112358, 1321345, 5891442)
AND         A.ORDER_DATE > '2005/06/30'
ORDER BY    A.ORDER_DATE;

CREATE INDEX ORDERS_IX3a ON ORDERS (CUSTOMER_NO, ORDER_DATE);
CREATE INDEX ORDERS_IX3b ON ORDERS (ORDER_DATE, CUSTOMER_NO);

-- Query 4
SELECT      A.CUSTOMER_NO, A.CUSTOMER, A.ORDER_DATE
FROM        ORDERS A
WHERE       A.CUSTOMER_NO = 0112358
            OR  A.ORDER_DATE = '2005/06/30';

CREATE INDEX ORDERS_IX4 ON ORDERS (CUSTOMER_NO);
CREATE ENCODED VECTOR INDEX ORDERS_EVI4
        ON ORDERS (ORDER_DATE);
```

Indexing Strategy - Examples

```
-- Query 5
SELECT      A.CUSTOMER_NO, B.CUSTOMER, A.ORDER_DATE, A.QUANTITY
FROM        ORDERS A,
            CUSTOMERS B,
            ITEMS C
WHERE       A.CUSTKEY = B.CUSTKEY
AND         A.ITEMKEY = C.ITEMKEY
AND         A.CUSTOMER_NO = 0112358;

CREATE INDEX ORDERS_IX5a ON ORDERS (CUSTOMER_NO, CUSTKEY);
CREATE INDEX ORDERS_IX5b ON ORDERS (CUSTOMER_NO, ITEMKEY);
CREATE INDEX CUSTOMERS_IX5 ON CUSTOMERS (CUSTKEY);
CREATE INDEX ITEMS_IX5 ON ITEMS (ITEMKEY);
```


Indexing Strategy – EVI INCLUDE example (7.1)

```
-- Query 6
SELECT      YEAR(A.ORDER_DATE),SUM(A.QUANTITY), COUNT(*)
FROM        ORDERS A
GROUP BY    YEAR(A.ORDER_DATE);

CREATE ENCODED VECTOR INDEX ORDERS_IX6A
        ON ORDERS (YEAR(ORDER_DATE))
        INCLUDE (SUM(QUANTITY), COUNT(*));
```

Indexing Strategy - Examples

-- Query 7

```
SELECT     YEAR(A.ORDER_DATE),QUARTER(A.ORDER_DATE),
           MONTH(ORDER_DATE), SUM(A.QUANTITY), COUNT(*)
FROM       ORDERS A
WHERE      QUARTER(A.ORDER_DATE) = 4
GROUP BY   YEAR(A.ORDER_DATE), QUARTER(A.ORDER_DATE),
           MONTH(ORDER_DATE)
ORDER BY   YEAR(A.ORDER_DATE),QUARTER(A.ORDER_DATE),
           MONTH(ORDER_DATE),

CREATE ENCODED VECTOR INDEX ORDERS_IX6A
      ON ORDERS (YEAR(ORDER_DATE), QUARTER(A.ORDER_DATE),
      MONTH(ORDER_DATE) )
      INCLUDE (SUM(QUANTITY), COUNT(*));
```


Indexing Strategy - Examples

If the optimizer feedback indicates:

Full table scan → Create an index on local selection columns

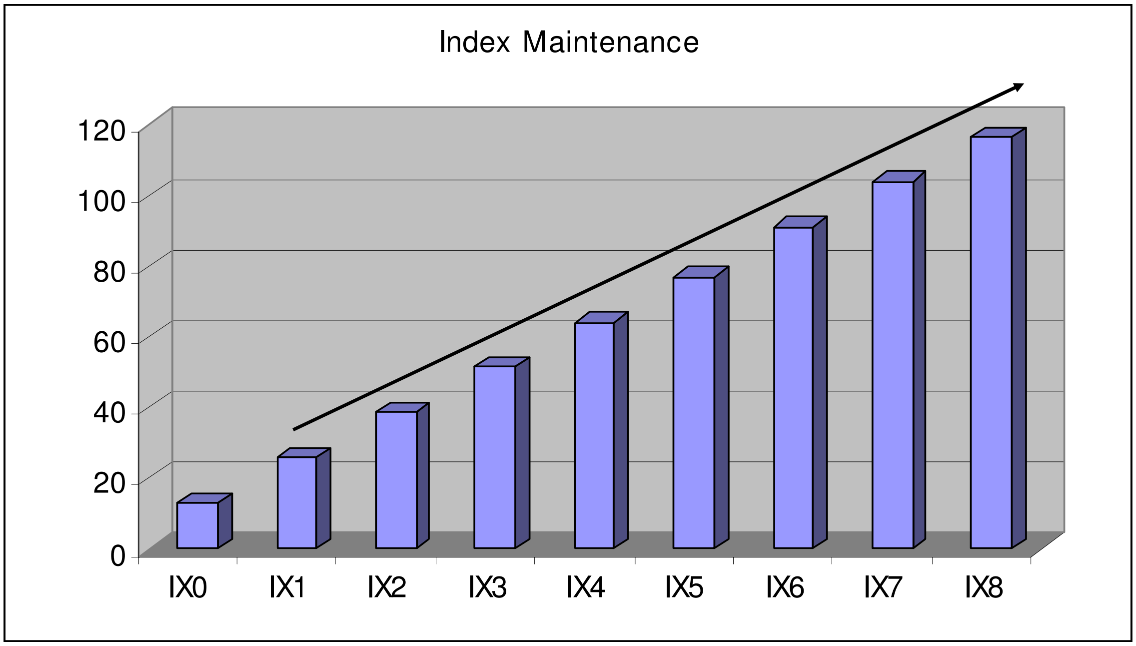
Full index scan → Create an index that allows probe

Temporary index → Create an index on join columns
→ Create an index on grouping columns
→ Create an index on ordering columns

Hash table → Create an index on join columns
→ Create an index on grouping columns

“Perfect”, multiple key column radix indexes are usually best

Indexing Strategy – Maintenance



In general - index maintenance costs grows linearly

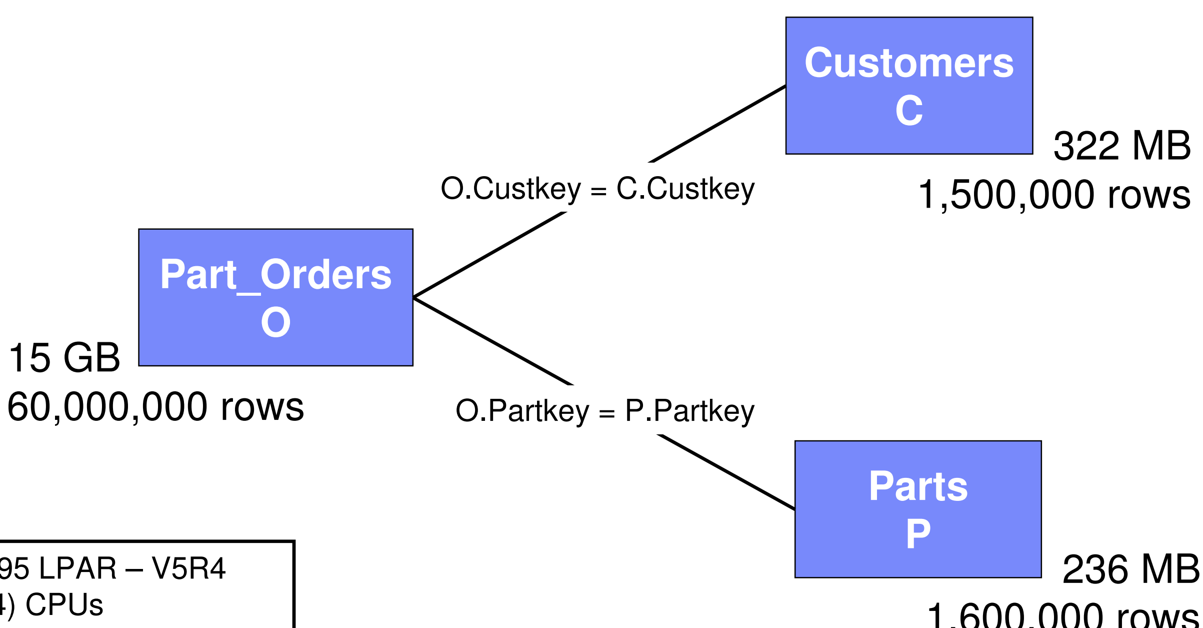
Indexing Strategy – Maintenance v Query Access

- **For best query performance, create the appropriate indexes**
- **Eliminating table scans and temporary data structures will more than make up for index maintenance overhead**

- Consider the number of indexes when doing *high* volume batch operations
- Consider parallel index maintenance for INSERTs
  - DB2 SMP feature installed and enabled
- Drop indexes when inserting into an empty table
- Consider dropping indexes when adding, changing or deleting more than 50% of the rows
  - Use SMP to create indexes in parallel
  - (INSERT + INDEX CREATION) < (INSERT + INDEX MAINT)

Indexing Case Study

Indexing Strategy – Case Study



595 LPAR – V5R4
(4) CPUs
10 GB in memory pool
(45) 70 GB disk units

Indexing Strategy – Case Study

- 80 SQL requests from a single JDBC connection…
  - 2 SETs
  - 53 SELECTs
  - 15 INSERTs
  - 5 UPDATEs
  - 15 DELETEs
  - 73 via SQE
  - 5 via CQE

- Scenarios…
  1. No indexes
  2. Indexes on join columns only
     - 4 radix indexes
  3. Indexes for selecting, joining, grouping, ordering
     - 13 radix indexes
     - 2 encoded vector indexes

Indexing Strategy – Case Study

– Indexes on join columns only

```
✓create index part_orders_ix1   on part_orders   (custkey);
✓create index part_orders_ix2   on part_orders   (partkey);
✓create index customers_ix1     on customers     (custkey);
✓create index parts_ix1         on parts         (partkey);
```

– Index for selecting, joining, grouping, ordering

```
✓create index part_orders_ix3   on part_orders   (returnflag, custkey);
✓create index part_orders_ix4   on part_orders   (shipmode, custkey);
✓create index part_orders_ix5   on part_orders   (orderkey, linenumber, custkey);
✓create index part_orders_ix6   on part_orders   (orderkey, custkey);
✓create index part_orders_ix7   on part_orders   (returnflag, partkey);
✓create index part_orders_ix8   on part_orders   (shipmode, partkey);
✓create index part_orders_ix9   on part_orders   (orderkey, linenumber, partkey);
✓create index customers_ix2     on customers     (customer, custkey);
✓create index parts_ix2         on parts         (part, partkey);

✓create encoded vector index part_orders_evi1 on part_orders   (returnflag);
✓create encoded vector index part_orders_evi2 on part_orders   (shipmode);
```

Indexing Strategy – Case Study

– Sample of SQL Requests

- select *
- from part_orders
- where custkey = 1
- and orderkey = 303008;

Highly Selective

- select *
- from part_orders o, customers c
- where o.custkey = c.custkey
- and c.customer = 'Customer#000000001';

Highly Selective
2way Join

- select *
- from part_orders o, customers c, parts p
- where o.custkey = c.custkey
- and o.partkey = p.partkey
- and c.customer = 'Customer#000000001'
- and o.orderkey = 303008
- order by o.linenumber;

Highly Selective
3way Join
Ordering

Indexing Strategy – Case Study

– Sample of SQL Requests

- select distinct shipmode
- from part_orders
- order by shipmode;

No Local Selection
Distinct
Ordering

- select shipmode, count(*)
- from part_orders
- group by shipmode
- order by 2 desc;

No Local Selection
Grouping
Ordering

Indexing Strategy – Case Study

– Sample of SQL Requests

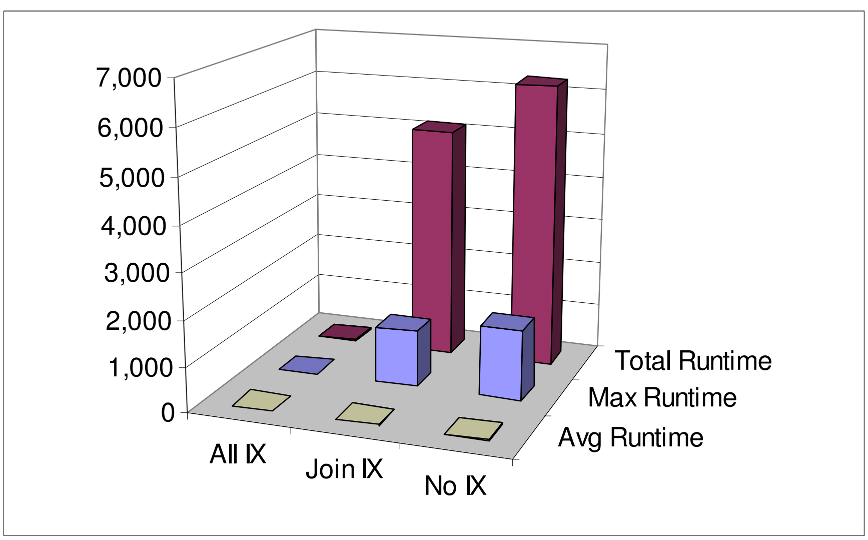
- update part_orders set expander = 'UPDATED'
- where custkey = 1;
- delete from part_orders
- where custkey = 1;
- insert into part_orders
- select * from item_subset1;

Searched Update
Highly Selective

Searched Delete
Highly Selective

Select Small Set
And Insert

Indexing Strategy – Case Study Results



	Total Time	Max Time	Avg Time
All Indexes	23.547	2.493	0.076
Join Indexes	5,138.851	1,249.081	20.975
No Indexes	6,302.275	1,533.910	20.265

Indexing Strategy – Case Study Results

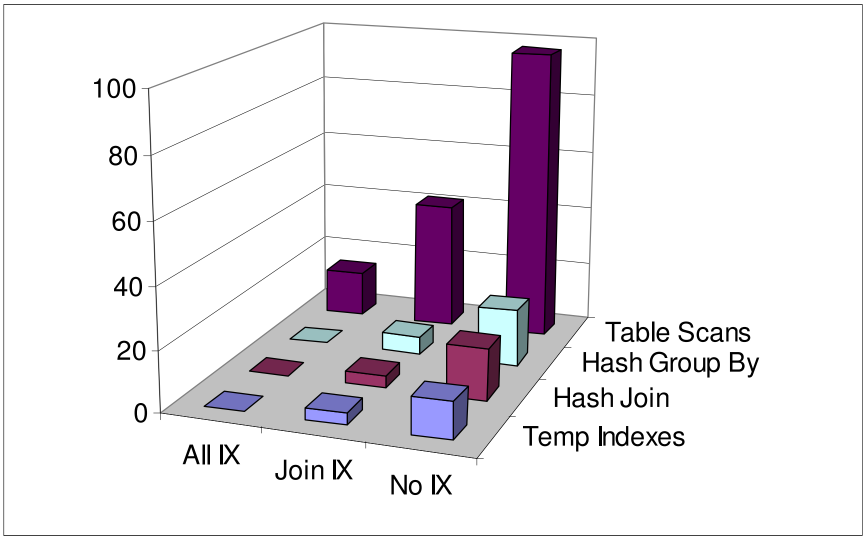
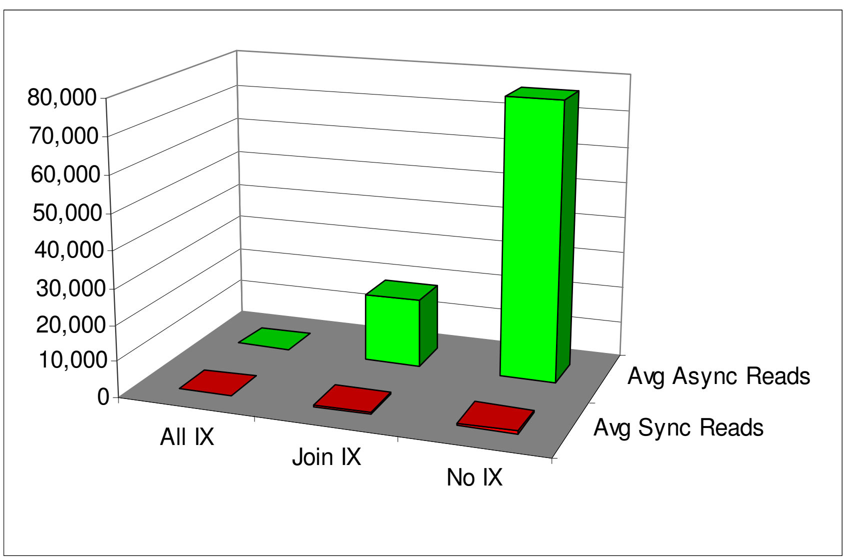


	Table Scans	Hash GroupBy	Hash Join	Temp Indexes
All Indexes	15	0	0	0
Join Indexes	42	6	4	4
No Indexes	97	19	17	12

IBM Power Systems

Indexing Strategy – Case Study Results



	Avg Async Reads	Avg Sync Reads
All Indexes	15	0
Join Indexes	42	6
No Indexes	97	19

Additional Information

- Indexing and Statistics strategies Whitepaper
  - http://www-304.ibm.com/partnerworld/wps/servlet/ContentHandler/servers/enable/site/bi/strategy/index.html
  - This Paper was updated to include 7.1 content.

- DB2 for i SQL & Query Performance Tuning and Monitoring Workshop
  - http://www-03.ibm.com/systems/i/software/db2/db2performance.html

- Text Search indexing technology
  - https://www-304.ibm.com/partnerworld/wps/servlet/ContentHandler?contentId=a7QzcNSQBe_4MDADcnt&roadMapId=IbOtoNReUYN4MDADrdm&roadMapName=Education+resources+for+IBM+i+systems&locale=en_US

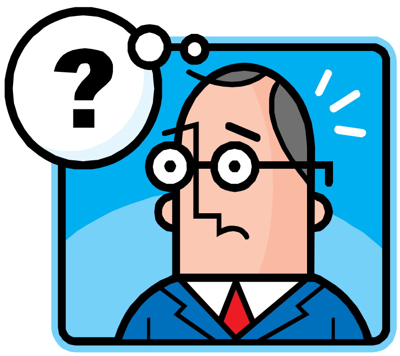
Summary

- Understand the technology and tools behind indexes on DB2 for i

- Monitor, analyze and create the most beneficial indexes
  - Its an iterative process – data can change, queries can change
  - Don't just create. Drop indexes that are not used

- The right set of indexes can:
  - significant improve performance of your application
  - improve overall system health

- **Are you experiencing performance problems?**
- **Are you using SQL?**
- **Are you getting the most out of DB2 for i?**



Need help?

IBM DB2 for i Center of Excellence

- ✓ Database modernization
- ✓ DB2 Web Query
- ✓ Database architecture and design
- ✓ DB2 SQL performance analysis and tuning
- ✓ Data warehousing and Business Intelligence
- ✓ DB2 for i education and training

Contact: Tom McKinley mac2@us.ibm.com
IBM Systems and Technology Group
Rochester, MN USA

Thank You

Trademarks and Disclaimers

Adobe, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Cell Broadband Engine and Cell/B.E. are trademarks of Sony Computer Entertainment, Inc., in the United States, other countries, or both and are used under license therefrom.

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

Other company, product, or service names may be trademarks or service marks of others.

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

The customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

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

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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

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

Prices are suggested U.S. list prices and are subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.