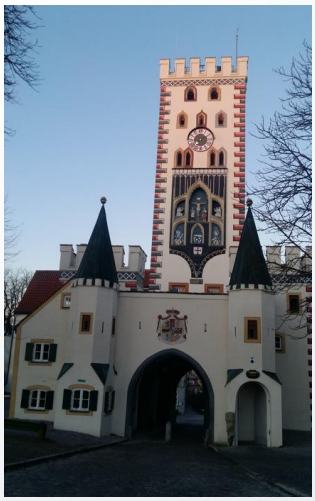
IBM i Access Client Solutions (ACS)
SQL Performance Analysis (Introduction) – Visual Explain

Birgitta Hauser –
Modernization – Education – Consulting on IBM i
Diplom-Betriebswirt (BA)
Database and Software Architect
Hauser@ModEdCon.com
https://modedcon.com/

QUSER
Twin Cities IBM Power Systems User Group

IBM Champion since 2020

Landsberg am Lech - Bayertor



QUSER
Twin Cities IBM Power Systems User Group

IBM Champion since 2020

Agenda

Query Process

Performance Analysis Feedback Tools

Access Path

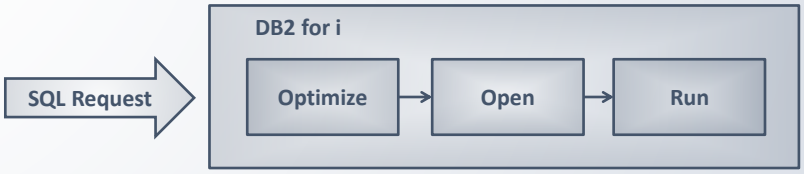
- DDS described logical Files
- Binary Radix Tree Indexes
- EVI Encoded Vector Indices

Visual Explain

- Works with Visual Explain
- Detail Information
- Index Advise
- Examples

Query Process

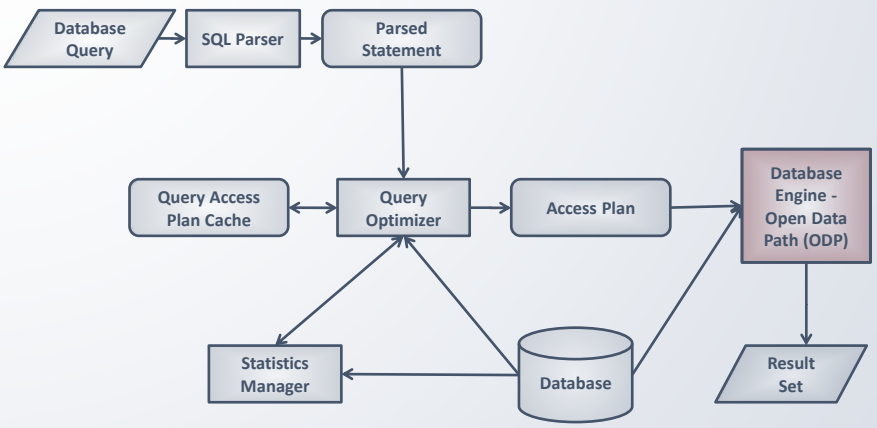
Executing an SQL Statement - New Request



Query Optimizer
- Determines the **best** method to fulfill the SQL request
  - Based on the existing **access paths** (indexes/DDS logical files) and based on the **composition** of the data
    → Information is provided by the **Statistics Manager**

- Provides the recipe and operating instructions (access plan) but does **NO** cooking

Executing an SQL Statement



- Most time consuming part of the query optimization

Access Plan

Each Query needs an Access Plan to be executed

Access Plan is the Output of the Query Optimization

- Access Plan is built or validated during the Query Optimization

Access Plan is the Description how the Query must be executed
→ NOT the Execution itself

Access Plan Content

- Objects to be accessed: Table / Binary Radix Tree Indexes and Encoded Vector Indexes (EVI) incl. MTI (temporary Indexes)
- Access Method: Scan / Probe
- Sequence: in which the Database Objects must be accessed
- Temporary Objects: Hash table / Sorted List / Values List / Bitmap …

FULL OPEN and HARD CLOSE

FULL OPEN

- **Access Plan** has to be created or validated
  - Based on the existing access paths (SQL indexes and DDS logical files)
  - In composition with the information provided by the statistics manager
- **Open Data Path (ODP)** has to be established (based on the access plan)
  - **Most time consuming part** when executing a query

HARD CLOSE

- **Open Data Path** (ODP) is **deleted** → **Next run: FULL OPEN** necessary
  - Temporary structures described in the access plan are deleted

PSEUDO CLOSE

- **Open Data Path** (ODP) is **not deleted** → **Next run: Data** is only **refreshed**
  - Temporary structures remain existent, i.e. ODP is reuseable

PSEUDO OPEN

- **Open Data Path** (ODP) can be **reused** → **Next run: No Optimization** required

Executing a Query - Query Phases

Query Validation
- Validate query request (syntax checking/rewriting/adding information)
- Validate existing access plan (e.g. SQE Plan Cache, SQL Package)

Query Dispatcher
- Determine the query engine (CQE or SQE) that performs the query

Query Optimization
- Choose the most efficient method
- Build the access plan

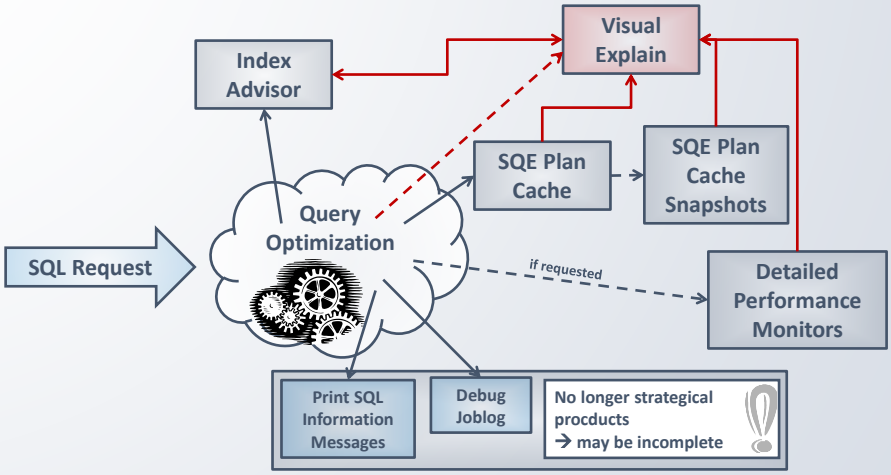
We can affect this
- **Access Paths / Indexing Strategy**
- **Syntax used in the SQL statement**

Query Execution
- Build temporary structures and/or indexes / Open the data path
- Monitors and adapts (AQP – Adaptive Query Processing – 7.1)
- **Gathers query information**
  - Index Advices / Access Plans (SQE Plan Cache)
  - Database monitor information (if requested)

Access Client Solution (ACS)
provides interfaces for analyzing these information

Performance Analysis Feedback Tools Overview

Query Optimization Feedback



Access Client Solution (ACS) - SQL – Performance Analyse

Access Client Solution (ACS) – Visual Explain

- **Graphically** displays the **Access Plan** for a SQL statement
  - Select, Insert, Update and Delete statements are supported
  - Shows access paths (Indexes / LF) and access methods to be used
  - Identifies the most time consuming steps
  - Provides Index and Statistics Advices

Access Client Solution (ACS) – SQE Plan Cache Analysis

- Interface for analyzing the **access plan information** for all **SQL statements** executed by the **SQE (SQL Query Engine)** gathered since the last IPL.
- **Disadvantage:** Queries executed by the CQE are not considered

CL /Access Client Solution (ACS) – Database/Performance Monitor

- **Gathers on demand information** about **all SQL statements** (even Query/400) independent of the Query Engine
- **Analysis:** SQL Queries / Query/400
  Access Client Solution (ACS)– **Performance Monitor Interfaces**

Visual Explain

Visual Explain

Part of IBM i Access Client Solutions (ACS)

- Currently **no equivalent** for the **Web Interface**

Visualization of the Access Plan

- Displays the permanently and temporarily used **Database Objects**
  - **Access Methods** on permanent Database Objects
  - **Temporary** Objects (e.g. Hash Tables, Temporary Lists)
  - **Dependencies** between permanent and temporary Database Objects
- **Execution Sequence**
- **Highlights:**
  - **Index Advices** → Display/Creation of the adviced Indexes
  - Used **Materialized Query Tables** (MQT)
  - Icons for **most expensive part** (dependent on time oder number of rows)
  - **Look Ahead Predicate Generation** (LPG)

Visual Explain

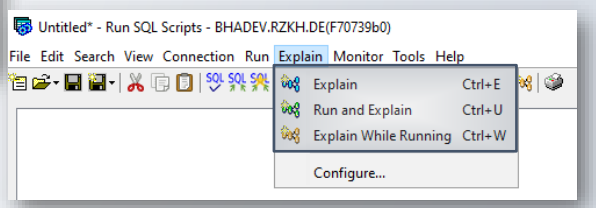
Detailed Information about the Access Plan

- **Job and Environment Information**
  - System, Job, Job Environment
  - Memory, Memory Pool Id / Size, Number of Processors, Parallel Degree
- **Query Engine**
- **Statement Informationen**
  - Use of the QAQQINI Query Options File incl. the Options (individually) set
  - Reusable ODP, Pseudo/Hard Close
- **Detailed Information** for each Node (Icon)
  - Table, Base Tabelle, Index, temporary Objects (e.g. Hash Table)
  - Durations (e.g. Runtime, Optimization Time, Opening Time, Fetch Time)
  - Number of processed Rows

Based on SQLStatement

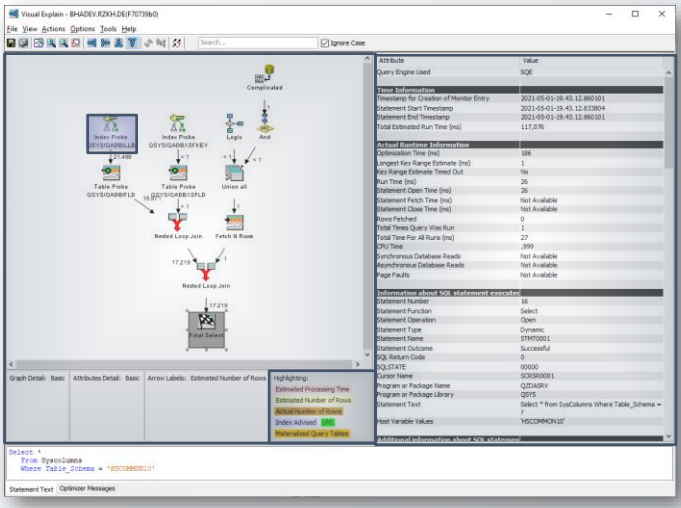
- If no appropriate access path exists, it will be **created at runtime**

Access Client Solution (ACS) - Run SQL Script – Visual Explain



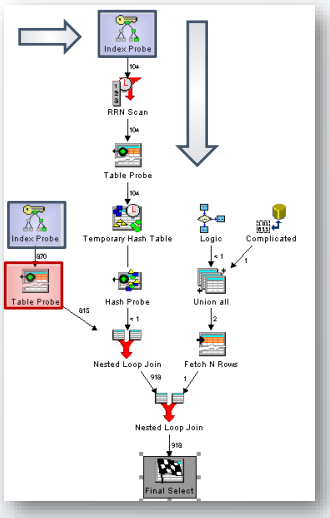
- **Position on the SQL Statement**
- **Run VisualExplain**
  - **Click on Icons**
  - **Select from Pulldown Menue**
- **Explain:** Explain without running
- **Run and Explain:** Execute and explain the query
- **Explain While Running:** Explain an already running query

Access Client Solution (ACS) - Visual Explain



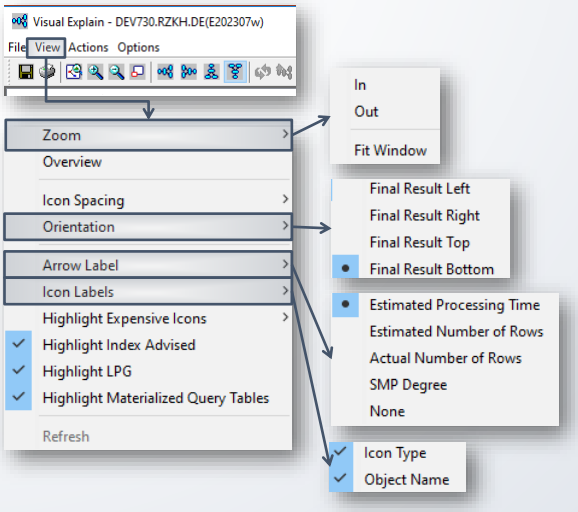
- Graphical Representation of the Access Plan
- Additional detailed Information
- Legend what is highlighted in which color
- Index Advices available

Visual-Explain – Graphical Representation Access Plan



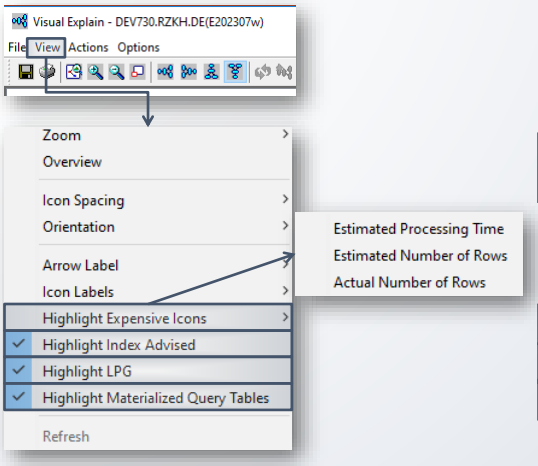
- Access Methods
  e.g. Index Probe, Table Scan
- Permanent Objects
  e.g. Tables, Indexes, Functions
- Temporary Objects
  e.g. temporary Hash Table
- Actions
  e.g. Table Probe, Nested Loop Join
- Sequence
- Effort (Time / Number of Rows)
- Highlight adviced Indexes
- Highlight most expensive Operation

Visual Explain - View



- Distances between the icons
- Orientation of the graphical access path
- Additional Information to be displayed
- Display Icon Type and/or Object name

Visual Explain - View

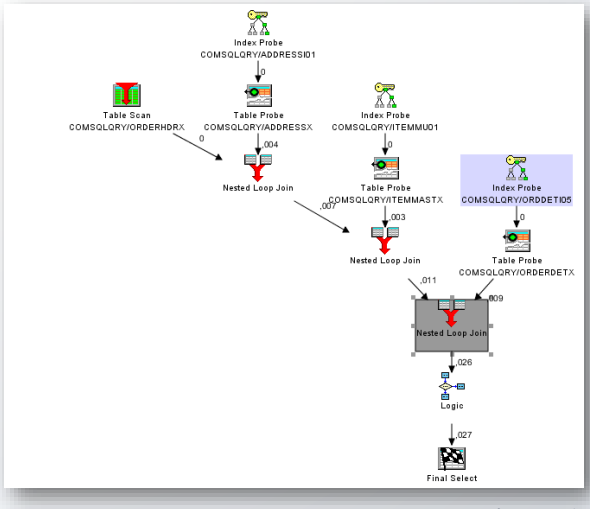


- Most expensive step → pink
- Depending on the time and/or number of rows

- Tables with advised indexes → blue
- Look Ahead Predicate Generation (LPG) → green
- MQT access → orange

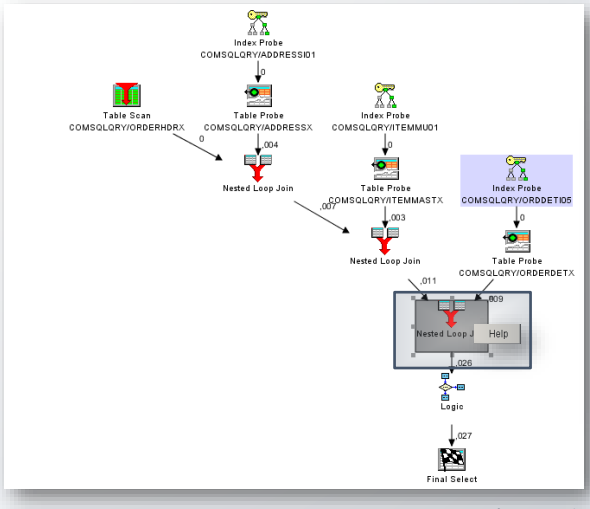
Work with Visual Explain

Visual Explain – Access Plan

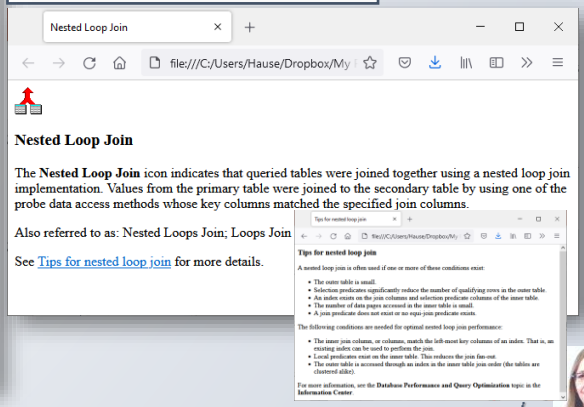


- Objects
- Structures
- Attributes
- Access Methods
- Actions
- Sequence / Flow
- Logic
- Costs / Estimates
- Strategy Methods

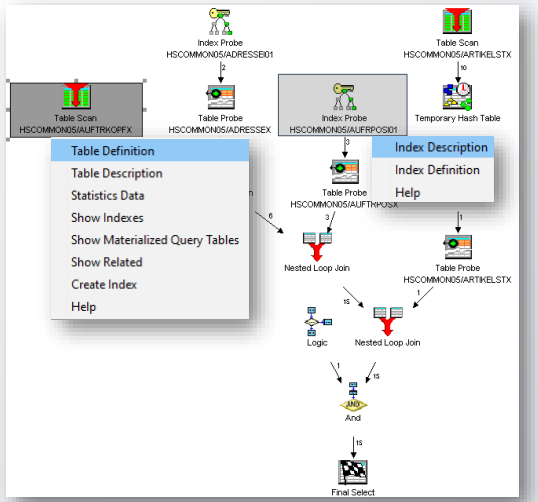
Visual Explain – Access Plan - Online Help



- Online Help for each icon
  - Position on the icon
  - Right click → Help



Visual Explain – Access Plan - Right click on Icons



- Detailed Information for the selected object
- Different Options depending on the Object Type/Icon

- **Attention**: Object must exist on the System

Visual Explain – Access Plan - Mouse Over → Attributes

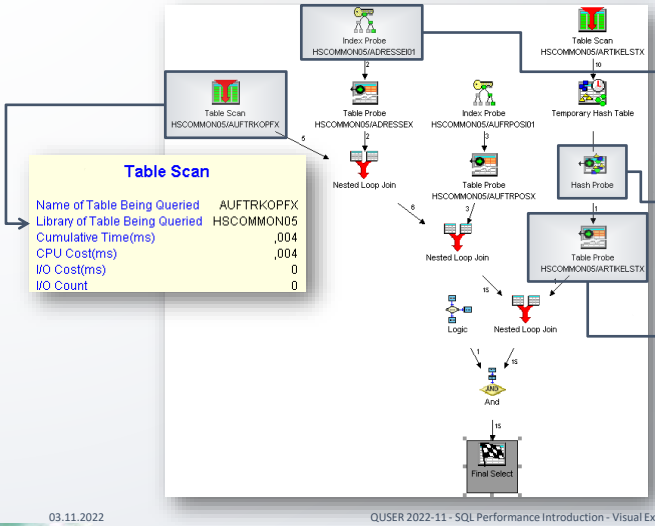


Table Scan

Name of Table Being Queried	AUFTRKOPFX
Library of Table Being Queried	HSCOMMON05
Cumulative Time(ms)	,004
CPU Cost(ms)	,004
I/O Cost(ms)	0
I/O Count	0

Index Probe

Name of Index Used	ADRESSEI01
Library of Index Used	HSCOMMON05
Cumulative Time(ms)	,004
CPU Cost(ms)	,004
I/O Cost(ms)	0
I/O Count	0

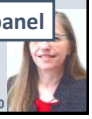
Hash Probe

Cumulative Time(ms)	,262
CPU Cost(ms)	,003
I/O Cost(ms)	0
I/O Count	0

Table Probe

Name of Table Being Queried	ARTIKELSTX
Library of Table Being Queried	HSCOMMON05
Cumulative Time(ms)	,263
CPU Cost(ms)	,001
I/O Cost(ms)	0
I/O Count	0

- Same information as displayed in the right panel



Visual Explain – Access Plan - Mouse Over → Attributes

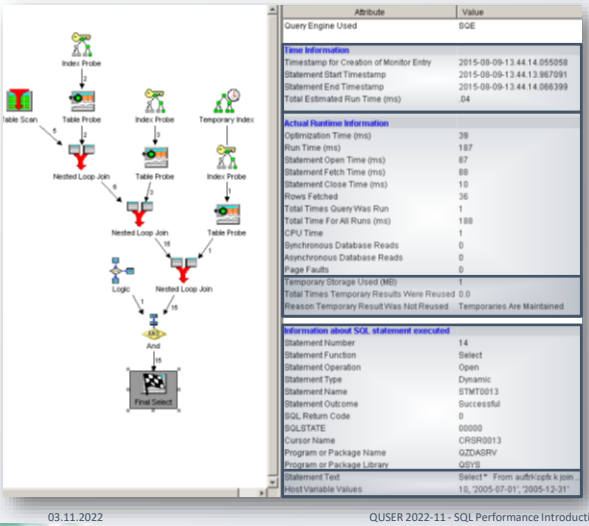
Table Probe

Name of Table Being Queried	ARTIKELSTX
Library of Table Being Queried	HSCOMMON05
Cumulative Time(ms)	,263
CPU Cost(ms)	,001
I/O Cost(ms)	0
I/O Count	0

- Cumulative Time (ms): accumulated time until this Icon
- CPU Cost (ms)
- I/O Costs
- I/O Count: Time spend for this step

Visual Explain - Detail Information

Visual Explain – Detail-Information - Final Select

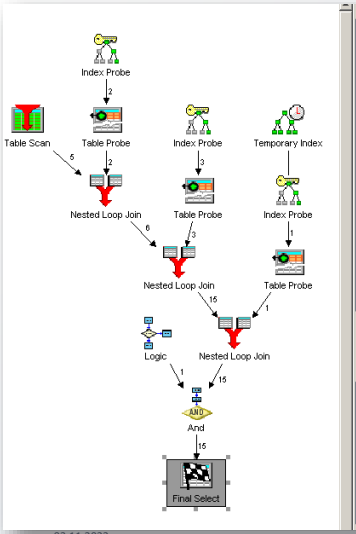


Attribute	Value
Query Engine Used	SQE
Time Information	
Timestamp for Creation of Monitor Entry	2015-08-09-13.44.14.055058
Statement Start Timestamp	2015-08-09-13.44.13.967091
Statement End Timestamp	2015-08-09-13.44.14.066399
Total Estimated Run Time (ms)	.04
Actual Runtime Information	
Optimization Time (ms)	39
Run Time (ms)	187
Statement Open Time (ms)	87
Statement Fetch Time (ms)	88
Statement Close Time (ms)	10
Rows Fetched	36
Total Times Query Was Run	1
Total Time For All Runs (ms)	188
CPU Time	1
Synchronous Database Reads	0
Asynchronous Database Reads	0
Page Faults	0
Temporary Storage Used (MB)	1
Total Times Temporary Results Were Reused	0.0
Reason Temporary Result Was Not Reused	Temporaries Are Maintained
Information about SQL statement executed	
Statement Number	14
Statement Function	Select
Statement Operation	Open
Statement Type	Dynamic
Statement Name	STMT0013
Statement Outcome	Successful
SQL Return Code	0
SQLSTATE	00000
Cursor Name	CRSR0013
Program or Package Name	QZDASRV
Program or Package Library	QSYS
Statement Text	Select * From auftrkopfk k join
Host Variable Values	10, '2005-07-01', '2005-12-31'

- Information about
  - Optimize
  - Open
  - Run
- Total Time
- Runtime
- MTI Usage
- SQL Statement
  - Statement text
  - Host variables

Note: Double click on statement
→ complete Statement Text visible

Visual Explain – Detail-Information - Final Select

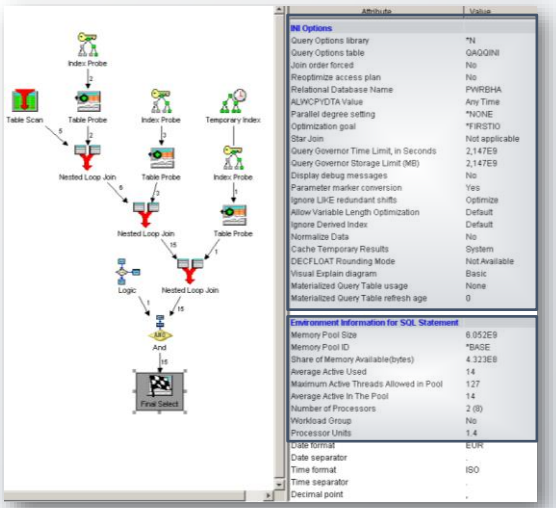


Attribute	Value
Additional information about SQL statement	
CLOSQLCSR Value	
ALWCPYDTA Value	Any Time
Pseudo Open	No
Pseudo Close	No
Hard Close Reason Code	Not Available
ODP Implementation	Reusable
Dynamic Replan Reason Code	The query options have changed
Timestamp When Plan Was Created	2015-08-09-13.44.13.977148
Data Conversion Reason Code	Not applicable
Blocking Enabled	ALWBLK(*ALLREAD)
Delay Prep	Yes
Statement is Explainable	Yes
Naming Convention	System
Type of Dynamic Processing	Local Prepared Statement
SQL Path	QSYS2924 ,QSYS ,QSYS2
Concurrent Access Resolution Used	Not applicable
IP Port Number	8,471
Client IP Address	192.168.7.9
IP Address Type	1
XML data CCSID	1,208
AQP Used in Access Plan	No
AQP Access Plan Iteration	1
Information common to most monitor rows	
System Name	PWRBHA
Job Name	QZDASOINIT
Job User	QUSER
Job Number	512368
Current User	BHA
Unique Query Count	85
Refresh Count	0

- Additional Statement-Information
  - Hard Close / Pseudo Close/Open
  - Reuseable/ ODP

- Job / User Information

Visual Explain – Detail-Information - Final Select

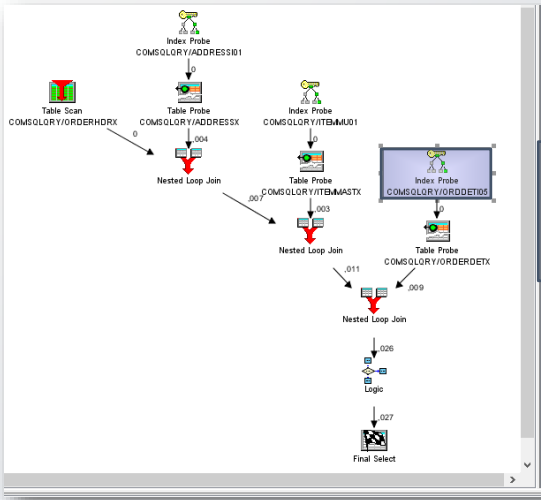


Attribute	Value
INI Options	
Query Options library	*N
Query Options table	QAQQINI
Join order forced	No
Reoptimize access plan	No
Relational Database Name	PWRBHA
ALWCPYDTA Value	Any Time
Parallel degree setting	*NONE
Optimization goal	*FIRSTIO
Star Join	Not applicable
Query Governor Time Limit, in Seconds	2,147E9
Query Governor Storage Limit (MB)	2,147E9
Display debug messages	No
Parameter marker conversion	Yes
Ignore LIKE redundant shifts	Optimize
Allow Variable Length Optimization	Default
Ignore Derived Index	Default
Normalize Data	No
Cache Temporary Results	System
DECFLOAT Rounding Mode	Not Available
Visual Explain diagram	Basic
Materialized Query Table usage	None
Materialized Query Table refresh age	0
Environment Information for SQL Statement	
Memory Pool Size	6.052E9
Memory Pool ID	*BASE
Share of Memory Available(bytes)	4.323E8
Average Active Used	14
Maximum Active Threads Allowed in Pool	127
Average Active In The Pool	14
Number of Processors	2 (8)
Workload Group	No
Processor Units	1.4
Date format	EUR
Date separator	
Time format	ISO
Time separator	
Decimal point	

- QAQQINI Query Options File – Options
  - Optimization Goal
  - Ignore Derived Index
  - Cache temporary Results
  - MQT Usage

- Environment
  - Memory Pool Size
  - Memory Pool Id
  - Number of Processors

Visual Explain – Access Plan - Highlight - Index Advised

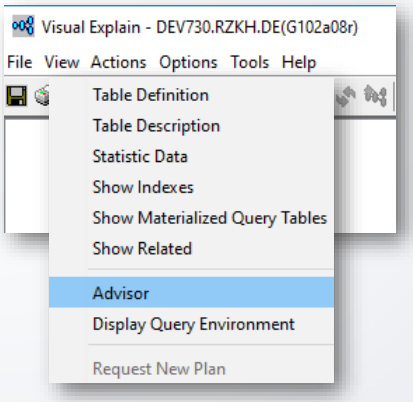


I/O Cost(ms)	0
I/O Count	2,372
Average IO Read Time (ms)	1,001
I/O Cost Reduction	Yes
PreLoad Relation	No
Memory Used(bytes)	155,489
Memory Constrained	No
Cumulative Memory Constrained	No
Index advised information	
Creation of an Index is Advised	Yes
Number of Primary Key Columns	3
Library of Base Table	COMSQLQRY
Name of Base Table	ORDERDETX
List of Key Columns for Advised Index	ITEMNO,COMPANY,ORDERNO
Type of Index Created	BINARY RADIX
ACS Table Name	*HEX
ACS Table Library	*N
Actual Runtime Information	
Actual Rows Selected Per Plan Step Iteration	3
Actual Rows Processed Per Plan Step Iteration	4
Actual Plan Step Iterations	6
Actual Total Rows Selected	18
Actual Total Rows Processed	22
Information About the Plan Performed	
Contains Predicate	Yes
Scrollable	Yes
Plan Name	IndexProbe
Plan Step Type	Logic
Plan Step Name	Node_604
Statement Text	SELECT DseIdField(ORDDETI05) FROM Radix Index(COMSQLQRY/ORDDETI05) WHERE

- Tables / Indexes with adviced indexes highlighted in blue
- Detailed Information about the adviced Index i.a. Number of Keys, List of Key Columns in the recommended Sequence

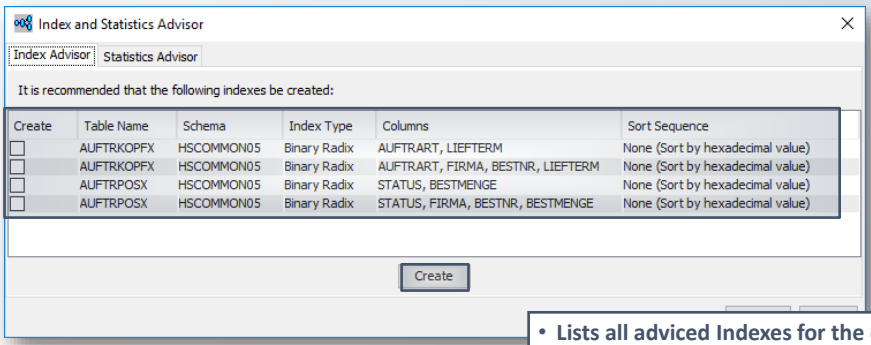
Visual Explain – Pull Down Menu Index and Statistics Advice

Visual-Explain – Pull-Down Menu - Actions



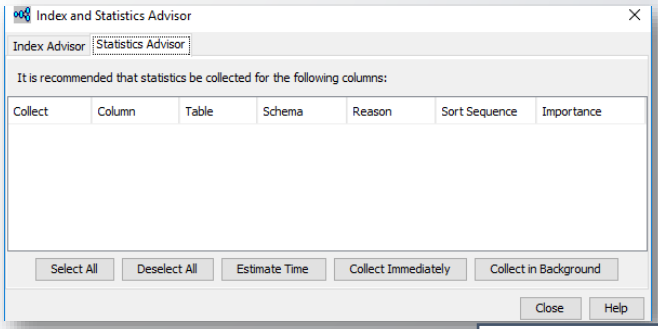
- Information about all used permanent Database Objects
  - Tables
  - Indexes
  - Materialized Query Tables (MQT)
  - User Defined (Table) Functions (UD(T)F)
  - Dependend Objects
  - Statistic Data
- Index and Statistics Advisor
  - Listing all adviced Indexes
    → Option for to generate the Indexes directly
  - Listing missing Statistics
    → Option for updating the Statistics

Visual-Explain – Pull-Down Menu – Actions - Index-Advisor



- Lists all adviced Indexes for the current query
  - Table, Schema
  - Index Type
  - Key Fields
  - Sort Sequence
- Create Indices by clicking on the Create-Button

Visual-Explain – Pull-Down Menu – Actions - Statistics-Advisor



- Lists all for the query needed but currently not updated Statistics

- Buttons:
  - Select All: Select all Statistics for to be updated
  - Deselect All: Reset all selected Statistics
  - Estimate Time: Time estimation for updating the Statistics
  - Collect Immediately:
  - Collect in Backround: Update in Batch

Access Path – Logical Files

Logical Files

DDS described logical files

SQL provides **2** different types of logical files

- **View** Can be specified in **SQL Statements**
  Can be accessed **with native I/O** (but are never keyed)
- **Index** Is only used by the **Query Optimizer**
  but **cannot** be specified within **SQL Statements**
  Can be accessed **with native I/O**
  like any keyed DDS described file

DDS Logical Files versus SQL Indexes and Views

	DDS Terminology		SQL Terminology			
	Logical File		View		Index Binary Radix Index	
Object Type	*FILE		*FILE		*FILE	
Attribute	LF		LF		LF	
Name	System Name	Up to **10 characters**	Long name	Up to **128 characters**		
			System Name	System Name = SQL Name if shorter than 10 characters		
				Can be specified in the **CREATE**-Statement (Release 7.1 TR6) or after creation by using the SQL command **RENAME**		
				Automatically created if the SQL name exeeds 10 characters		
Creation	CL Command	**CRTLF** (Create Logical File)	SQL Command	**CREATE VIEW**	SQL Command	**CREATE INDEX**
Logial Page Size	Per Default between **8** and **32 K**				Per Default **64 K**	
Format	Multiple format logical files		Format of the base table will be inherited Keyword **RCDFMT** for views and indexes **Only a single format** allowed			

DDS Logical Files versus - SQL Indexes and Views

	DDS Terminology	SQL Terminology	
	Logical File	View	Index Binary Radix Index
Order	Key Fields		All Key Information is stored in indexes
Field Selection	List Fields	SELECT Colum1, Column2, ... ColumnN	Derived Indexes - Field selection
	Generating additional Columns	Using artithmetic operators, concatenating strings, using scalar and user defined functions	
Row Selection	SELECT/OMIT clause	WHERE Conditions	Sparse Indexes - WHERE Conditions allowed
Joining Files and Tables	DDS Keywords (JOIN, JFILE, JFLD)	INNER JOIN / FULL OUTER JOIN LEFT / RIGHT OUTER JOIN LEFT / RIGHT EXCEPTION JOIN CROSS JOIN	
		Can be used in other views (like any table)	
Consolidation of rows		GROUP BY - Clause	
Use of Functions	A few keywords for example SST (substring)	All scalar functions Use of global Variables All aggregat functions in composition with GROUP-BY User Defined Functions (UDF) User Defined Table Functions (UDTF) incl. XMLTABLE Common Table Expressions incl. Recursive CTE Hierachical Query Clause Nested Sub-Selects UNION / INTERSECT / EXCEPT	Derived Indexes - the use of scalar functions and basic arithmetic operations (+, -, *, /) and character concatenation is allowed

DDS described logical Files

DDS described logical Files

Based on either DDS described physical Files or SQL Tables

- **Do not include any data** but an description how to access these data

Keyed (logical) Files

- Include an **access path** for accessing the data in a **pre-defined sequence**

Field/Column Selections

- **All** or a **sub-set** of the **fields/columns** in the base physical file(s)/table(s)
- **Additional fields/columns** in conjunction with predefined **keywords** (e.g. SST)

Row Selection

- **Select / Omit** clauses

Joined (keyed) logical files

- **Multiple physical files** and/or SQL tables can be joined together
- **Key Columns** only from **one** of the joined physical files

No equivalent in SQL for
Keyed Joined and multi-format LF

Format

- **Single** or **Multiple format** logical files can be defined

Accessing DDS described logical Files

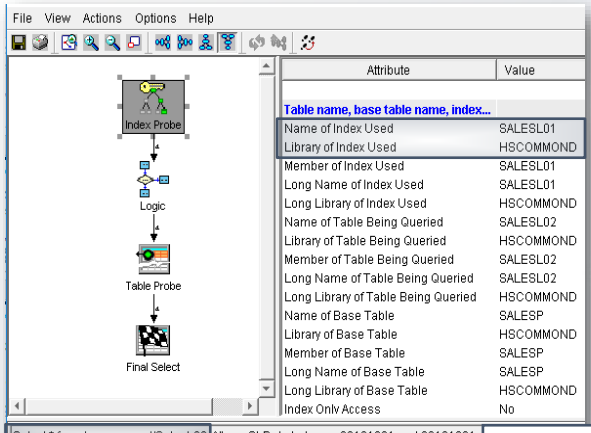
Native I/O (RPG and Cobol)

- **Traditional way:** for positioning/chaining a **single row/record** or accessing **data** in a **predefined sequence**
- Logical files are **directly specified** within the **source code** → **specified** logical files **are used**
  → **No optimization** required

SQL

- **SQL Statement:** DDS described logical files **can be specified** but **should not** be used
  → **Query optimization**
- **Query Optimizer:** **rewrites** the SQL Statement based on the **physical files/SQL tables**
  decides after if any / which access paths are used

DDS described logicals File and SQL




```
Select *
  from SalesL02
  Where SLDate between 20131201 and 20131231;
```

```
A          R SALESF                    PFILE(SALESP)
 *
 * KEY FIELDS
A          K SLDATE    R
A          K SLCUST    R
A          K SLITEM    R
```

- SALESL01

```
A          R SALESF                    PFILE(SALESP)
 *
 * KEY FIELDS
A          K SLCUST    R
A          K SLDATE    R
A          K SLITEM    R
```

- SALESL02

- Logical file SALESL02 was specified but SALESL01 is used
- SQL/**Query Optimizer decides** if any or which access path is used

Indexes

Index – Keyed Logical File

Keyed logical File

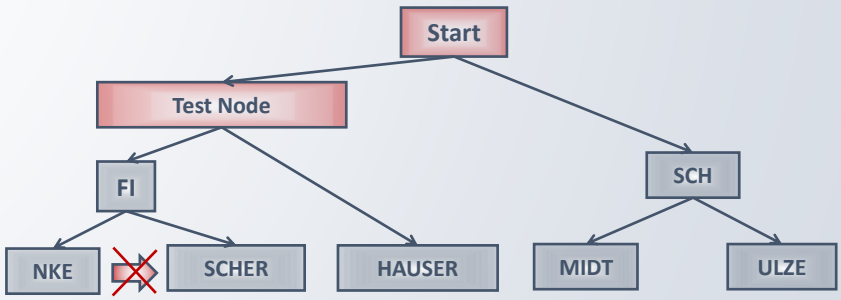
- Creation with the SQL Command: **`CREATE INDEX`**

Index Types

- **Binary Radix Tree Index** → can be compared with a DDS described keyed logical file built over a single physical file/SQL Table
- **Encoded Vector Index (EVI)** → IBM Patent
- **Omnifind Text Search Index** → Special Index (stored in IFS)
  perform high-speed linguistic text searches against character data
  Licence Product – 5733-OMF Replaces Db2 Text Extender

Binary Radix Index

Database Table			
RRN	Key Field(s)	Additional Field(s)	
1	SCHULZE		
2	FISCHER		
3	HAUSER		
4	SCHMIDT		
5	FINKE		
6	...		
...	...		

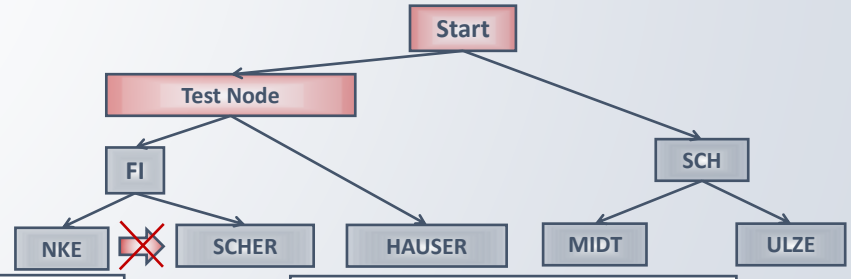


Advantages:
- Quick access to a single key value
  1 Mill. Records ~ 20 Trials
- Also efficient for small, selected range of key values (**low cardinality**)

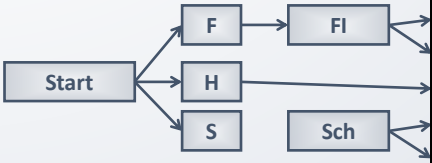
Disadvantages:
- Table rows retrieved in order of key values (not physical order) which equates to many *RANDOM* I/Os when selecting a large number of keys (**high cardinality**)
- **No way** to predict which physical index pages are next when traversing the index for large number of key values

Binary Radix Index

Database Table		
RRN	Key Field(s)	Additional Field(s)
1	SCHULZE	
2	FISCHER	
3	HAUSER	
4	SCHMIDT	
5	FINKE	
6	…	
…	…	



Binary Radix Tree – Key Information



Bitmap – Row/Record/Data Locations

FINKE	0	0	0	0	**1**	0	0	0	0	0	0	0	0	…
FISCHER	0	**1**	0	0	0	0	0	0	0	0	0	0	0	…
HAUSER	0	0	**1**	0	0	0	0	0	0	0	0	0	0	…
SCHMIDT	0	0	0	**1**	0	0	0	0	0	0	0	0	0	…
SCHULZE	**1**	0	0	0	0	0	0	0	0	0	0	0	0	…

Enhanced Indexing Technologies

Derived Index

- **Additional Key Columns** can be defined
  - Basic **arithmetic operations** (+, -, *, /) on several columns
  - **Concatenating** character columns (concat) – Except LOBs (Large Objects)
  - Usage of SQL **scalar functions** (for example **UPPER, YEAR, SUBSTR**)

```
Create Index HSCOMMON10.SalesDI01
    on HSCOMMON10.Sales
      (Year(SalesDate) as SalesYear ASC, CustNo, ItemNo)
```

Sparse Index

- **WHERE Conditions** for restricting Data Access can be added
  - Similar to SELECT/OMIT clauses in logical files, but **much more powerful**
  - **Logical Operators** (AND/OR/NOT), **scalar Functions, Parenthesis**

```
Create Index HSCOMMOND.ADDRESSX_I04 For System Name ADDRI04
    On ADDRESSX (COUNTRY, CITY, CUSTNAME1, CUSTNAME2, STREET)
       Where      COUNTRY In ('D' , 'A' , 'CH')
            And Lower(Trim(CUSTNAME1) CONCAT ' ' CONCAT Trim(CUSTNAME2)) Like '%gmbh%'
Rcdfmt ADDRESSXF Add All COLUMNS;
```


Derived Indexes - Example

```
Select City, CustName1, CustName2, Street,
       a.*
  From AddressX a
  Order By City, CustName1, CustName2;;
```

CITY	CUSTNAME1
	Gartengeräte Hübner
frankfurt	eugen fischer gmbh
Achern	Haustechnik Spengler
Alzenau	Alzenauer Dönertreff
Alzenau	Goldbach GmbH
Alzenau	Top Videothek
Apfelstadt	Fruits, Vegetables & Co
Aschaffenburg	Radio NAMISLO & Partner
Augsburg	Media Markt
Augsburg	Musicstore GmbH
ALZENAU	MARKUS HAEFNER GMBH
ALZENAU	SCHNEIDER AG
Berlin	Alfred Mayer Gartenzubehör K
BERLIN	ARBURG GMBH
Den Haag	Philips AG
Dietzenbach	25 % Discount
Dortmund	LIDL AG

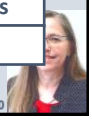
- All text columns in different **upper, lower** and **mixed case** notation

```
Create Index Hscommon10.Addressx_I03 For System Name ADDRI03
    on ADDRESSX
       (Country,
        Lower(Trim(City))       as CITY_LOWERCASE      For Column LowCity,
        Lower(Trim(CustName1))  as CUSTNAME1_LOWERCASE For Column LowName1,
        Lower(Trim(CustName2))  as CUSTNAME2_LOWERCASE For Column LowName2,
        Lower(Trim(Street))     as STREET_LOWERCASE    For Column LowStreet)
    RcdFmt ADDRESSXF
    Add All Columns;;
```

- SQL Name
- System Name

ADDRESS_I03 Index

- New key columns based on the LOWER and TRIM **scalar** functions
- **SQL** and **System** Column and Index Names

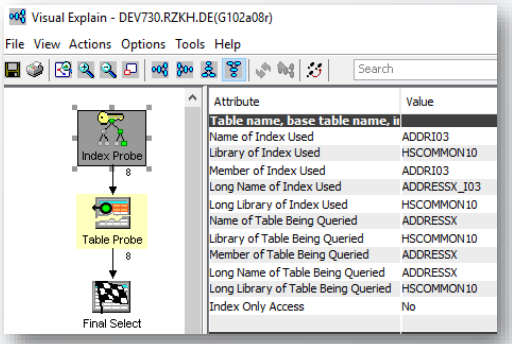


Derived Indexes and Query Optimizer

```
Create Index Hscommon10.Addressx_I03 For System Name ADDRI03
    on ADDRESSX
       (Country,
        Lower(Trim(City))       as CITY_LOWERCASE      For Column LowCity,
        Lower(Trim(CustName1))  as CUSTNAME1_LOWERCASE For Column LowName1,
        Lower(Trim(CustName2))  as CUSTNAME2_LOWERCASE For Column LowName2,
        Lower(Trim(Street))     as STREET_LOWERCASE    For Column LowStreet)
    RcdFmt ADDRESSXF
    Add All Columns;;
```

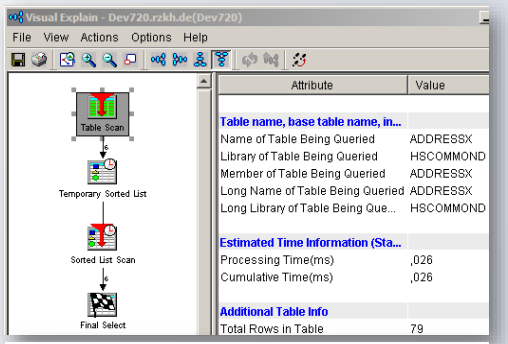
- Indexes **cannot** be specified in an SQL Statement → **Query Optimizer** decides!
- Index can only be used by the Query Optimizer
  if the SQL expressions specified in the SELECT-Statement **exactly match** the SQL expressions in the index definition

Derived Indexes and Query Optimizer




```
Select City, ZipCode, CustName1, CustName2
From AddressX
Where     Country = 'D'
      and Lower(Trim(City)) = 'frankfurt'
Order By Lower(Trim(City)), Lower(Trim(CustName1))
```

- Index Access: SQL Expressions match




```
Select City, ZipCode, CustName1, CustName2
From AddressX
Where     Country = 'D'
      and Trim(Lower(City)) = 'frankfurt'
Order By Trim(Lower(City)), Trim(Lower(CustName1))
```

- Table Scan: SQL Expressions do not match

Derived Indexes and Views

```
Create Index Hscommon10.Addressx_I03 For System Name ADDRI03
    on ADDRESSX
       (Country,
        Lower(Trim(City))       as CITY_LOWERCASE      For Column LowCity,
        Lower(Trim(CustName1))  as CUSTNAME1_LOWERCASE For Column LowName1,
        Lower(Trim(CustName2))  as CUSTNAME2_LOWERCASE For Column LowName2,
        Lower(Trim(Street))     as STREET_LOWERCASE    For Column LowStreet)
    RcdFmt ADDRESSXF
    Add All Columns;;
```

```
Create or Replace view HSCOMMON10.ADDRESSX_V03   For System Name ADDRV03
    as (Select a.*,
               Lower(Trim(City))       as LowCity,
               Lower(Trim(Custname1))  as LowName1,
               Lower(Trim(CustName2))  as LowName2,
               Lower(Trim(Street))     as LowStreet
          From AddressX a);;
```

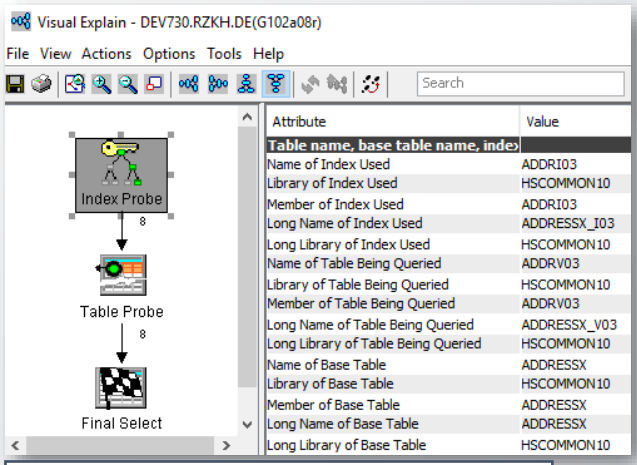
- Create and use **a view**
- Add **new** columns based on the excatly specified **expressions**

```
Select City, ZipCode, CustName1, custName2
   From AddrV03
   where Country = 'D' and LowCity = 'frankfurt'
   Order By LowCity, LowName1
```

- Reduced Source Code
- Masked complexity

- Business logic moved into the database

Derived Indexes and Views and Query Optimizer




```
Select City, ZipCode, CustName1, custName2
   From AddrV03
   where Country = 'D' and LowCity = 'frankfurt'
   Order By LowCity, LowName1
```

- Access View → Index Probe! Espressions match!

Sparse Indexes

Query Optimizer (not) using Sparse Indexes

```
Create Index ADDRESSX_I05   For System Name ADDRI05
  on ADDRESSX
    (COUNTRY,
     Lower(Trim(City))       as CITY_LOWERCASE      For Column LOWCITY,
     Lower(Trim(CustName1)) as CUSTNAME1_LOWERCASE For Column LOWNAME1,
     Lower(Trim(CustName2)) as CUSTNAME2_LOWERCASE For Column LOWNAME2,
     Lower(Trim(Street))     as STREET_LOWERCASE    For Column LOWSTREET)
     WHERE     COUNTRY in ('D', 'A', 'CH')
           and Lower(Trim(CUSTNAME1) concat
                     ' ' concat
                 Trim(CUSTNAME2))        like '%gmbh%'
  RcdFmt ADDRESSXF
  Add All Columns;
```

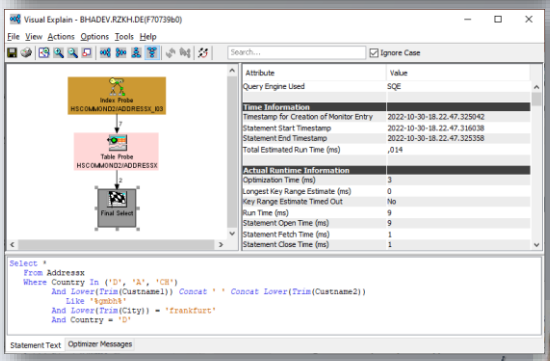
- Sparse Index with the following WHERE Conditions

```
WHERE     COUNTRY in ('D', 'A', 'CH')
      and Lower(Trim(CUSTNAME1) concat
                ' ' concat
            Trim(CUSTNAME2))        like '%gmbh%'
```

- Derived Index ADDRESSX_I03 is used!
- **Sparse Index ADDRESSX_I05 is not used**, because the values in the IN clause and in the LIKE predicate are considered as being **variable** and **not constant**!

- Executing a Query with exactly the same WHERE Conditions

```
Select *
  from Addressx
  WHERE     COUNTRY IN ('D', 'A', 'CH')
        AND LOWER(TRIM(CUSTNAME1)) CONCAT ' ' CONCAT LOWER(TRIM(CUSTNAME2)) LIKE '%gmbh%'
        And Lower(Trim(City)) = 'frankfurt'
        And Country = 'D';
```



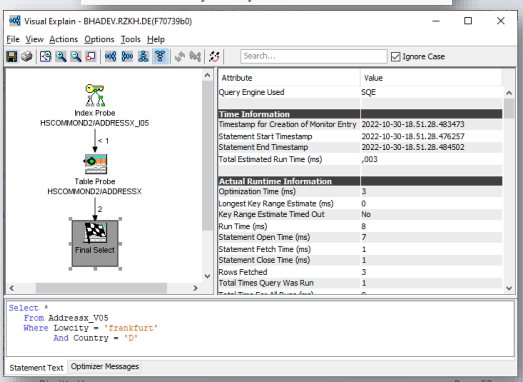
Query Optimizer (not) using Sparse Indexes

```
Create or Replace View ADDRESSX_V05   For System Name ADDRV05
    as (Select a.* ,
               Lower(Trim(City))       as LowCity,
               Lower(Trim(CustName1)) as LowName1,
               Lower(Trim(CustName2)) as LowName2,
               Lower(Trim(Street))     as LowStreet
          From ADDRESSX A
          Where     Country in ('D', 'A', 'CH')
                and Lower(Trim(CustName1) concat
                          ' ' concat
                      Trim(CustName2))        like '%gmbh%';
```

- Create an additional view with the same WHERE concitions as in the Index ADDRESSX_I03

- Run the same query as in the previous examples. Instead of accessing the table directly the view ADDRESSX_05 is used

```
Select *
  from Addressx_V05
  Where lowcity = 'frankfurt'
        and Country = 'D';
```



- **Sparse index ADDRESSX_I05 is used**
- The values in the IN clause and the LIKE predicate are now considered as **constant values!**

EVI - Encoded Vector Index

Encoded Vector Index

Index object for delivering fast data access in analytical query and reporting environments

- Advanced technology from IBM Research
- **Variation** on bitmap indexing
  - **Complement** to radix index
  - Easy to maintain
- Fast, easy access to **Satistics**
  - improve query optimizer decision making
- Used to **materialize** a **bitmap** or relative record number **(RRN) list**
  - In composition with **Star Join Schema** and **LPG** (Look Ahead Predicate Generation)

Include Aggregate Functions

Encoded Vector Index

Encoded Vector Index (EVI) Components

- **Symbol-Table**
  - Lists all **distinct key values**
  - Includes **additional statistical information** for each distinct key
    - Relative Record No. of the 1st/last occurence and Number of occurences
  - Assigns the **unique binary code** to the **distinct keys values**
    - Depending on the number of distinct keys → Code = 1, 2, 4 Byte
- **Vector**
  - Assigns the **unique code** to the **row/record** (Relative Record No)

```
CREATE Encoded Vector Index Myschema.MyEviName
    ON MySchema.MyTable
       (Col1 Asc/Desc, Expr1 Asc/Desc ...)
    With n Distinct Values
```


Encoded Vector Index

Symbol Table				
Key Value	Code	First Row	Last Row	Number of Rows
Fischer	1	2	80.005	53.207
Finke	2	5	99.760	7.586
…				
Hauser	7	3	10.032	1.764
…				
Schmidt	37	4	30.111	33.408
Schulze	38	1	83.000	27.607

Vector	
Row Number	Code
1	38
2	1
3	7
4	37
5	2
6	2
7	37
8	37
9	7
10	5
11	7
12	1

- Symbol table contains information for each distinct key value
- Each key value is assigned a unique code (key compression)
- Code is 1, 2, or 4 bytes depending on number of distinct key values
- Rather then a bit array for each distinct key value, use one array of codes

Encoded Vector Index

Symbol Table						
Key Value	Code	First Row	Last Row	Number of Rows	Sum()	Avg()
Fischer	1	2	80.005	53.207	1.234.567	23,2
Finke	2	5	99.760	7.586	222.333	29,3
…						
Hauser	7	3	10.032	1.764	98.877	56,2
…						
Schmidt	37	4	30.111	33.408	4.567.890	136,7
Schulze	38	1	83.000	27.607	234.099	8,5

Vector	
Row Number	Code
1	38
2	1
3	7
4	37
5	2
6	2
7	37
8	37
9	7
10	5
11	7
12	1

- Read Data from the Database

Encoded Vector Index - Enhancements

Include Clause

- Results of **aggregate functions** can be stored within the symbol table
  - `AVG, COUNT, COUNT_BIG, SUM, STDDEV, STDDEV_SAMP, VARIANCE, VARIANCE_SAMP`
- Enhanced to supports `CUBE, ROLLUP` and `GROUPING SETS` Enhanced RELEASE 7.2

Query Optimizer

- **Access methods**
  - EVI Symbol Table Scan
  - EVI Symbol Table Probe
- **Symbol Table Index Only Access**
  Aggregate information can be read out of the **symbol table directly**
  - No need to scan the vector
  - No need to access any row within the table

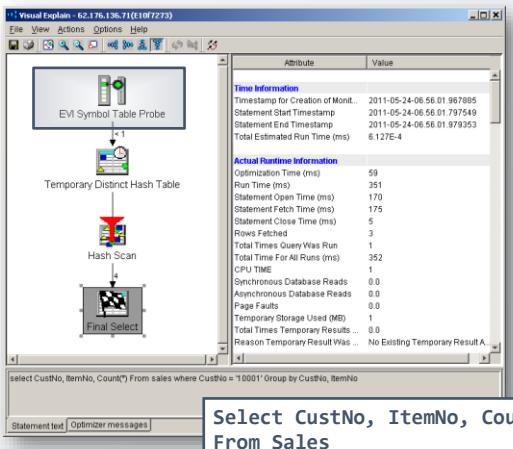
Encoded Vector Index

Symbol Table				
Key Value	Code	First Row	Last Row	Number of Rows
Fischer	1	2	80.005	53.207
Finke	2	5	99.760	7.586
…				
Hauser	7	3	10.032	1.764
…				
Schmidt	37	4	30.111	33.408
Schulze	38	1	83.000	27.607

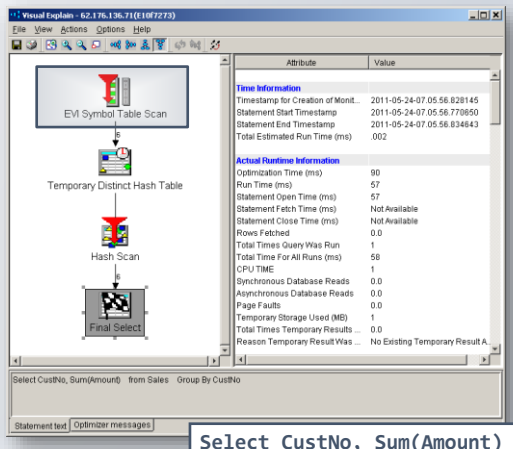
Vector	
Row Number	Code
1	38
2	1
3	7
4	37
5	2
6	2
7	37
8	37
9	7
10	5
11	7
12	1

- Symbol table contains information for each distinct key value
- Each key value is assigned a unique code (key compression)
- Code is 1, 2, or 4 bytes depending on number of distinct key values
- Rather then a bit array for each distinct key value, use one array of codes

Encoded Vector Index - Access




```
Select CustNo, ItemNo, Count(*)
From Sales
Where CustNo = '10001'
Group By CustNo, ItemNo
```




```
Select CustNo, Sum(Amount)
From Sales
Group By CustNo
```


Advanced Encoded Index Only Access

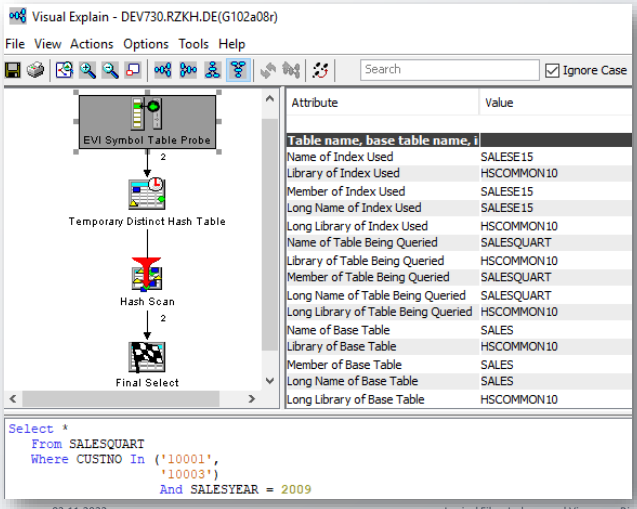
```
Create View HSCOMMON10.SalesQuart as
Select   CustNo,
         Year(SalesDate) as SalesYear,
         Cast(sum(case when Quarter(SalesDate)= 1 then Amount else 0 end) as Dec(11, 2)) as Q1,
         Cast(sum(case when Quarter(SalesDate)= 2 then Amount else 0 end) as Dec(11, 2)) as Q2,
         Cast(sum(case when Quarter(SalesDate)= 3 then Amount else 0 end) as Dec(11, 2)) as Q3,
         Cast(sum(case when Quarter(SalesDate)= 4 then Amount else 0 end) as Dec(11, 2)) as Q4,
         Cast(sum(Amount) as Dec(13, 2)) as Total
   from  Sales
   group by CustNo, Year(SalesDate);;
```

```
Select * from SalesQuart
Where CustNo in ('10001', '10003')
      and SalesYear = 2009
```

CUSTNO	SALESYEAR	Q1	Q2	Q3	Q4	TOTAL
10001...	2009	340,00	299,40	1118,50	876,30	2634,20
10003...	2009	733,00	1057,91	2227,45	571,50	4589,86

```
Create Or Replace Encoded Vector Index HSCOMMON10.SALESE15
    On HSCOMMON10.SALES (CUSTNO Asc, Year(SALESDATE) as SALESYEAR Asc)
    Include (Sum(Case When Quarter(SALESDATE) = 1 Then AMOUNT Else 0 End) ,
             Sum(Case When Quarter(SALESDATE) = 2 Then AMOUNT Else 0 End) ,
             Sum(Case When Quarter(SALESDATE) = 3 Then AMOUNT Else 0 End) ,
             Sum(Case When Quarter(SALESDATE) = 4 Then AMOUNT Else 0 End) ,
             Sum(AMOUNT));
```

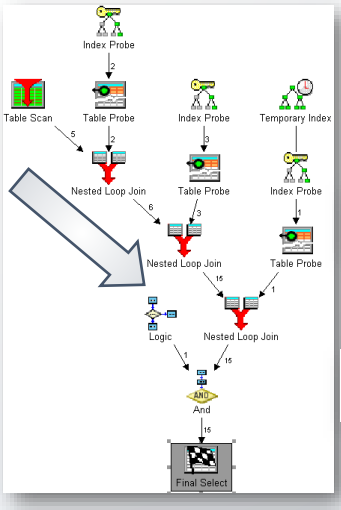

Advanced - Encoded Index Only Access




```
Select *
   From SALESQUART
   Where CUSTNO In ('10001',
                    '10003')
                   And SALESYEAR = 2009
```


Visual Explain - Examples

Visual Explain – Examples - Joining multiple Tables



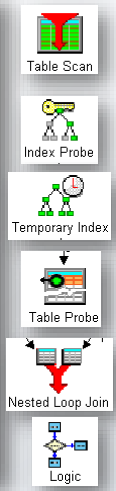


Table Scan
- Process all rows within a table (may cause Performance Issues)

Index Probe
- Direct access using the Index Key

Temporary Index
- Temporary Index (MTI) created/used

Table Probe
- Accessing the rows in composition with Indices, Values, RRN Lists

Nested Loop Join
- Joining the pre-selected information

Logic
- Executing additional logic for selecting data

Most efficient Access Method

There is more than one way to skin a cat …

- **Multiple Access Plans** for determining the **same result**
  - **Different time explosure**
  - **Different access methods** depending on the Number of occurrences, exsisting access path, data composition
- **Each Access Plan has its strengths and weaknesses**

Example: Table with `100,000,000` Rows / Column CUSTNO

- Select … `WHERE KUNDENR = 10001` …

Which method has to be used?

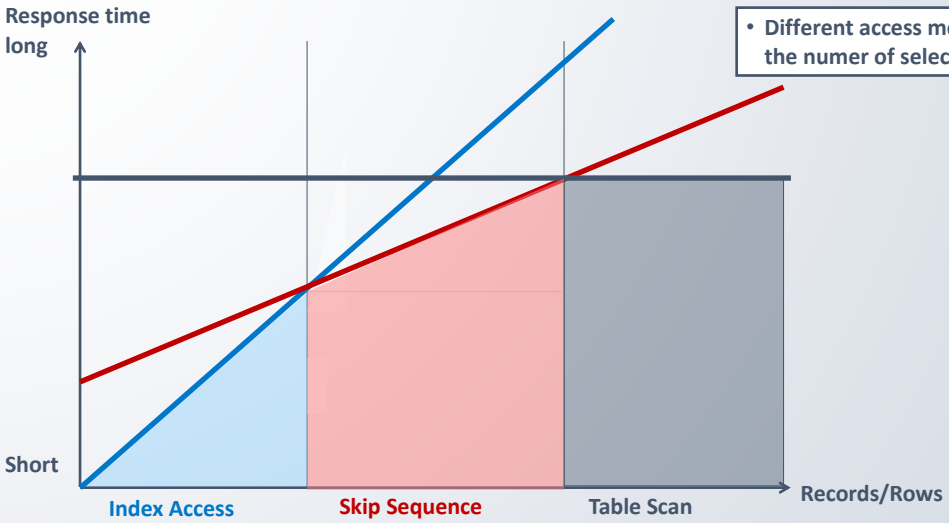
- If:

`1`	Row	with CustNo 10001
`1,000`	Rows	with CustNo 10001
`100,000`	Rows	with CustNo 10001
`100,000,000`	Rows	with CustNo 10001

- **Binary Radix Tree**
- **Binary Radix Tree**
- **Radix Tree or EVI**
- **Table Scan**

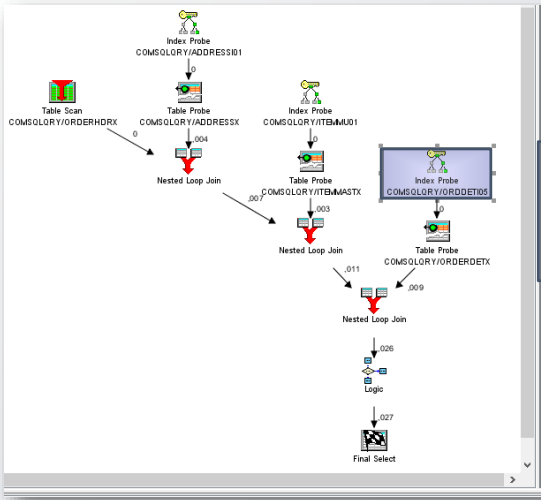
How the decision is made?

Access Methods



- **Different access methods depending on the numer of selected rows**

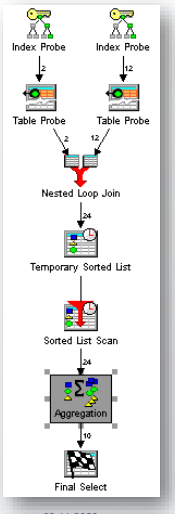
Visual Explain – Access Plan - Highlight - Index Advised

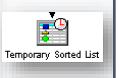


I/O Cost(ms)	0
I/O Count	2,372
Average IO Read Time (ms)	1,001
I/O Cost Reduction	Yes
PreLoad Relation	No
Memory Used(bytes)	155,489
Memory Constrained	No
Cumulative Memory Constrained	No
Index advised information	
Creation of an Index is Advised	Yes
Number of Primary Key Columns	3
Library of Base Table	COMSQLQRY
Name of Base Table	ORDERDETX
List of Key Columns for Advised Index	ITEMNO,COMPANY,ORDERNO
Type of Index Created	BINARY RADIX
ACS Table Name	*HEX
ACS Table Library	*N
Actual Runtime Information	
Actual Rows Selected Per Plan Step Iteration	3
Actual Rows Processed Per Plan Step Iteration	4
Actual Plan Step Iterations	6
Actual Total Rows Selected	18
Actual Total Rows Processed	22
Information About the Plan Performed	
Contains Predicate	Yes
Scrollable	Yes
Plan Name	IndexProbe
Plan Step Type	Logic
Plan Step Name	Node_604
Statement Text	SELECT DseIdField(ORDDETI05) FROM Radix Index(COMSQLQRY/ORDDETI05) WHERE

- Tables / Indexes with adviced indexes highlighted in blue
- Detailed Information about the adviced Index i.a. Number of Keys, List of Key Columns in the recommended Sequence

Visual Explain – Examples- Accumulation (GROUP BY)



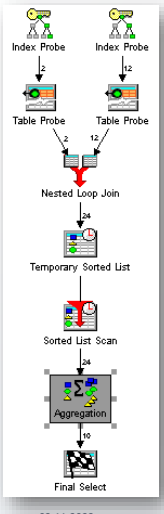


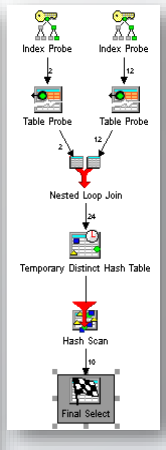
Temporary Sorted List
- Structure for gathering data and ordering them depending on a predefined sequence
  → GROUP BY

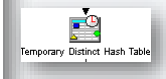
Sorted List Scan
- Process a temporary sorted list
  → ORDER BY

Aggregation
- Accumulate the gathered data

Visual Explain – Examples - Order By or Not to Order By







- Same Query: Difference → WITH and WITHOUT Order By

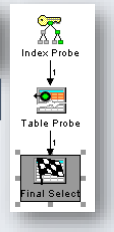
Temporary Distinct HashTable
- Hash Table with a single entry per group → Order By
- Sequence of the hashed data is not important

Hash Scan
- Scanning the Hash Table and returning the data

Visual Explain – Examples - Binary Radix Tree / EVI Index Access

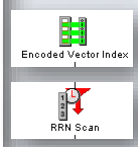
```
Select *
   From OrderHdrX
   Where Company = 10 and OrderNo = 'BNR2009-10-23/5';
```

- Index: Key Columns: OrderNo, Company



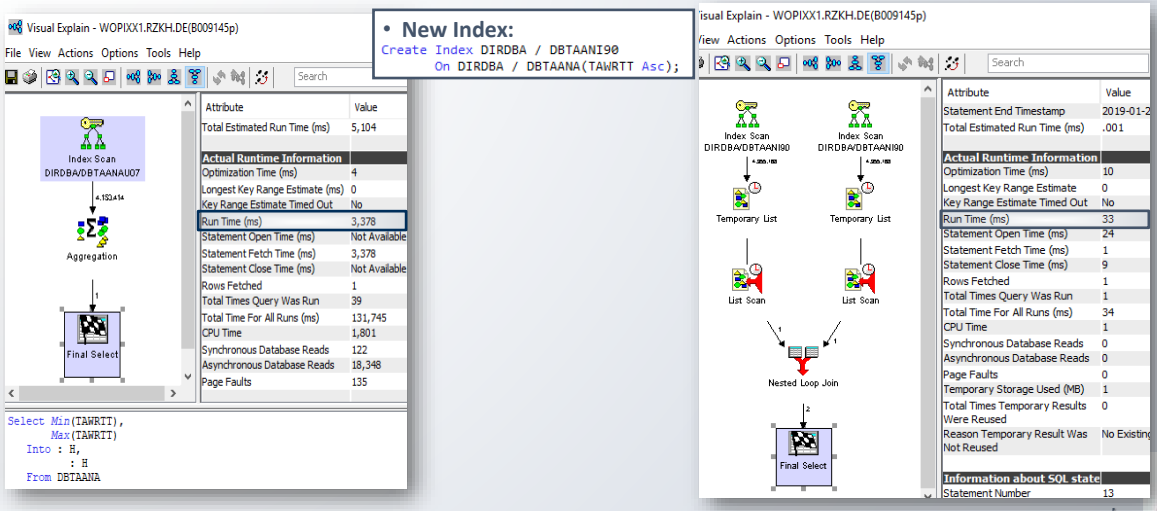

```
Select *
  From OrderHdrx
  Where Company = 10 and OrderType = 'DO'
  Order By Company, OrderNo;
```

- Encoded Vector Index (EVI)
- Keys Columns: Company, OrderType



- Reads the complete Encoded Vector Index (EVI) symbol table
- Scans the EVI vector for locating the selected rows

Optimization Index Scan



Visual Explain – Examples - Index Oring

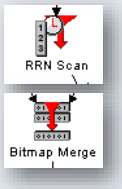
```
Select *
  From OrderHdrx
  Where     (    Company = 10
           and OrderNo = 'BNR2009-10-23/5')
        or  DelDate = '2009-12-01'
```

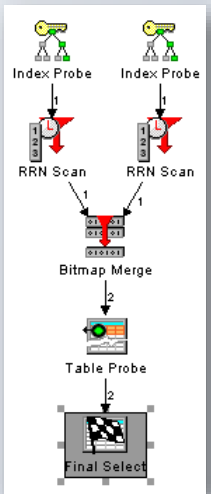
- Index 01: Key Columns: OrderNo, Company
- Index 02: Key Columns: DelDate

- Bitmaps containing a bit for each row which is set to on or off based on the key information

- Multiple Bitmaps are merged together

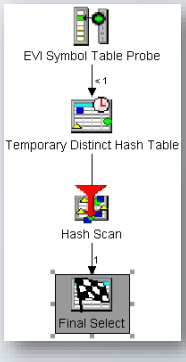
- Data access (Table Probe) occurs **after** based on the bitmap results





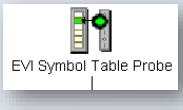
Visual Explain – Examples - Encoded Vector Access (EOA) - 1

```
Select year(SalesDate) SalesYear, CustNo,
       sum(Amount)
  from sales
  Where      Year(SalesDate) = '2009'
        and CustNO           = '10001'
  Group By year(SalesDate), CustNo
  Order By SalesYear, CustNo;
```



- EVI: Key Columns: CustNo, Year(SalesDate)
  INCLUDE Aggregate Information: Sum(Amount)
- All information to be returned is stored in
  - The **Key** information → Year(SalesDate), CustNo or in
  - The **Included aggregate** information → Sum(Amount)

→ **No need** to access any row (no Table Probe Icon)



- Keyed access on the EVI symbol table

Advanced Encoded Index Only Access

```
Create View HSCOMMON10.SalesQuart as
Select   CustNo,
         Year(SalesDate) as SalesYear,
         Cast(sum(case when Quarter(SalesDate)= 1 then Amount else 0 end) as Dec(11, 2)) as Q1,
         Cast(sum(case when Quarter(SalesDate)= 2 then Amount else 0 end) as Dec(11, 2)) as Q2,
         Cast(sum(case when Quarter(SalesDate)= 3 then Amount else 0 end) as Dec(11, 2)) as Q3,
         Cast(sum(case when Quarter(SalesDate)= 4 then Amount else 0 end) as Dec(11, 2)) as Q4,
         Cast(sum(Amount) as Dec(13, 2)) as Total
  from   Sales
  group by CustNo, Year(SalesDate);;
```

- View with Group By and Pivot Columns

```
Select * from SalesQuart
Where CustNo in ('10001', '10003')
      and SalesYear = 2009
```

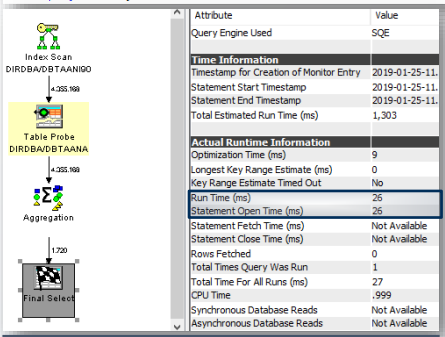
CUSTNO	SALESYEAR	Q1	Q2	Q3	Q4	TOTAL
10001...	2009	340,00	299,40	1118,50	876,30	2634,20
10003...	2009	733,00	1057,91	2227,45	571,50	4589,86

```
Create Or Replace Encoded Vector Index HSCOMMON10.SALESE15
    On HSCOMMON10.SALES (CUSTNO Asc, Year(SALESDATE) as SALESYEAR Asc)
    Include (Sum(Case When Quarter(SALESDATE) = 1 Then AMOUNT Else 0 End) ,
             Sum(Case When Quarter(SALESDATE) = 2 Then AMOUNT Else 0 End) ,
             Sum(Case When Quarter(SALESDATE) = 3 Then AMOUNT Else 0 End) ,
             Sum(Case When Quarter(SALESDATE) = 4 Then AMOUNT Else 0 End) ,
             Sum(AMOUNT));
```

- Encoded Vector Index (EVI)
  - Key Values: Year(SalesDate), CustNo
  - Include Clauses calculating Totals in Pivot Columns

EVIs with Includes

```
Select TAWRTT, Count(*),
               Sum(TAOINS), Sum(TAOUPD), Sum(TAODEL), Sum(TAOCLR), Sum(TAOCOP), Sum(TANBRR),
               Sum(TADSIZ), Sum(TAREAD), Sum(TADUSE), Sum(TAOOPN),
               Avg(TAOINS), Avg(TAOUPD), Avg(TAODEL), Avg(TAOCLR), Avg(TAOCOP), Avg(TANBRR),
               Avg(TADSIZ), Avg(TAREAD), Avg(TADUSE), Avg(TAOOPN)
  From DIRDBA.DBTAANA
  Group By TAWRTT;
```

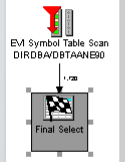


Attribute	Value
Query Engine Used	SQE
Time Information	
Timestamp for Creation of Monitor Entry	2019-01-25-11.
Statement Start Timestamp	2019-01-25-11.
Statement End Timestamp	2019-01-25-11.
Total Estimated Run Time (ms)	1,303
Actual Runtime Information	
Optimization Time (ms)	9
Longest Key Range Estimate (ms)	0
Key Range Estimate Timed Out	No
Run Time (ms)	26
Statement Open Time (ms)	26
Statement Fetch Time (ms)	Not Available
Statement Close Time (ms)	Not Available
Rows Fetched	0
Total Times Query Was Run	1
Total Time For All Runs (ms)	27
CPU Time	.999
Synchronous Database Reads	Not Available
Asynchronous Database Reads	Not Available

- **Binary Radix Tree Indexes only**

New EVI with Include

```
Create Encoded Vector Index DIRDBA .| DBTAANE90
       On DIRDBA / DBTAANA(TAWRTT Asc)
       Include(Count(*),
               Sum(TAOINS), Sum(TAOUPD), Sum(TAODEL), Sum(TAOCLR), Sum(TAOCOP),
               Sum(TANBRR), Sum(TADSIZ), Sum(TAREAD), Sum(TADUSE), Sum(TAOOPN),
               Avg(TAOINS), Avg(TAOUPD), Avg(TAODEL), Avg(TAOCLR), Avg(TAOCOP),
               Avg(TANBRR), Avg(TADSIZ), Avg(TAREAD), Avg(TADUSE), Avg(TAOOPN))
       With 65537 Distinct Values;
```



Attribute	Value
Statement End Timestamp	2019-01-25-12
Total Estimated Run Time (ms)	9.738
Actual Runtime Information	
Optimization Time (ms)	136
Longest Key Range Estimate (ms)	0
Key Range Estimate Timed Out	No
Run Time (ms)	13
Statement Open Time (ms)	13
Statement Fetch Time (ms)	Not Available
Statement Close Time (ms)	Not Available
Rows Fetched	0
Total Times Query Was Run	1
Total Time For All Runs (ms)	14
CPU Time	.999
Synchronous Database Reads	Not Available
Asynchronous Database Reads	Not Available

Tips for using Visual Explain

Check/Search for inefficient Access-Methods

- CQE Implentation
  - Determine/Revise reason → Execute with SQE
- **Table Scans** for Big Tables
  - Determine the portion of the rows to be processed
  - Create missing/adviced Indexes (Binary Radix Tree / Encoded Vector Indices)
- **Temporary Indexes**
  - Create permanent Indexes
- **Hash Tables**

Check Index and Statistics Advices

- Create adviced/necessary Indexes → rerun the query

Check Attributes of the current Environment

- Optimization Goal
- Memory Pool Size and Fair Share Calculation

Any Questions?

Special Thanks to

Holger Scherer – RZKH Rechenzentrum Kreuznach

- For providing an IBM i-System enabling the creation of the samples/code used in my presentations
- http://www.rzkh.de

RZKH RECHENZENTRUM KREUZNACH

Your data is save! … in the bunker

Speaker‘s Biography

Birgitta Hauser
Diplom-Betriebswirt (BA)
Database and Software Architect

Birgitta Hauser worked on the IBM i and its predecessors since 1992. She graduated with a business economics diploma, and started programming on the AS/400 in 1992. She worked and works as traditional RPG Programmer but also as Database and Software Engineer, focusing on IBM i application and database modernization.

Currently she is self-employed and works in Consulting and Application and Database Modernization on IBM i and Db2 for i. Since July, 2019 she is occasionally working for Fresche Solutions Inc. (Montréal) as a contractor.

She also works in education as a trainer for RPG and SQL developers.

Since 2002 she has frequently spoken at the COMMON User Groups and other IBM i and Power Conferences in Germany, other European Countries, USA and Canada.

In addition, she is co-author of two IBM Redbooks and also the author of several articles and papers focusing on RPG and SQL for the ITP Verlag (a German publisher), IT Jungle Guru and IBM DeveloperWorks.

In 2015 she received the John Earl Speaker Scholarship Award. In 2018 she received the Al Barsa Memorial Scholarship Award.

IBM Champion since 2020

Thank you!

SQL Performance Center and Visual Explain?

Yes i know!

If you are interested in more detailed individual Workshops on-site or remote, Please contact me directly

Birgitta Hauser – Modernization – Education – Consulting on IBM i
Diplom-Betriebswirt (BA)
Database and Software Architect
Hauser@ModEdCon.com
https://modedcon.com/