DB2 Platform: Unix, Windows, OS/2

Understanding DB2 Query Access Plans

IBM Canada Ltd
Ian Finlay
John Hornibrook
UDB Query Optimizer Development

May 3, 2000

Agenda

- The Explain Facility
  - What is it?
  - How to invoke it
  - Viewing options: Visual Explain and db2exfmt
  - What about db2expln and dynexpln?
- The Query Access Plan
  - Optimized SQL
  - Plan operators, arguments and properties
  - Predicate application
  - Execution flow (serial and parallel)
- Performance analysis tips

Confessions of a True Developer

- Development rarely uses the Explain facility as customers would
  - Preparing this presentation has been a learning experience
- Development has internal tools to complement the Explain facility
  - There is room for improvement with the Explain facility
  - some of the internal tools need to be externalized
  - new tools need to be developed
- This is not a performance tuning presentation

Agenda

- The Explain Facility
  - What is it?
  - How to invoke it
  - Viewing options: Visual Explain and db2exfmt
  - What about db2expln and dynexpln?
- The Query Access Plan
  - Optimized SQL
  - Plan operators, arguments and properties
  - Predicate application
  - Execution flow (serial and parallel)
- Performance analysis tips

The Explain Facility - What is it?

- Internal phase of the optimizer that captures critical information used in selecting the query access plan

- Two key external tools:
  - Explain tables with Visual Explain
    - Visual Explain offers a GUI interface to render and navigate query access plans
  - Explain tables with db2exfmt
    - db2exfmt offers a text-based output from the explain tables

- NOT db2expln or dynexpln
  - these are tools that interpret the runtime operators and generate limited information about the query access plan
  - not sufficient details about *why* operations where chosen

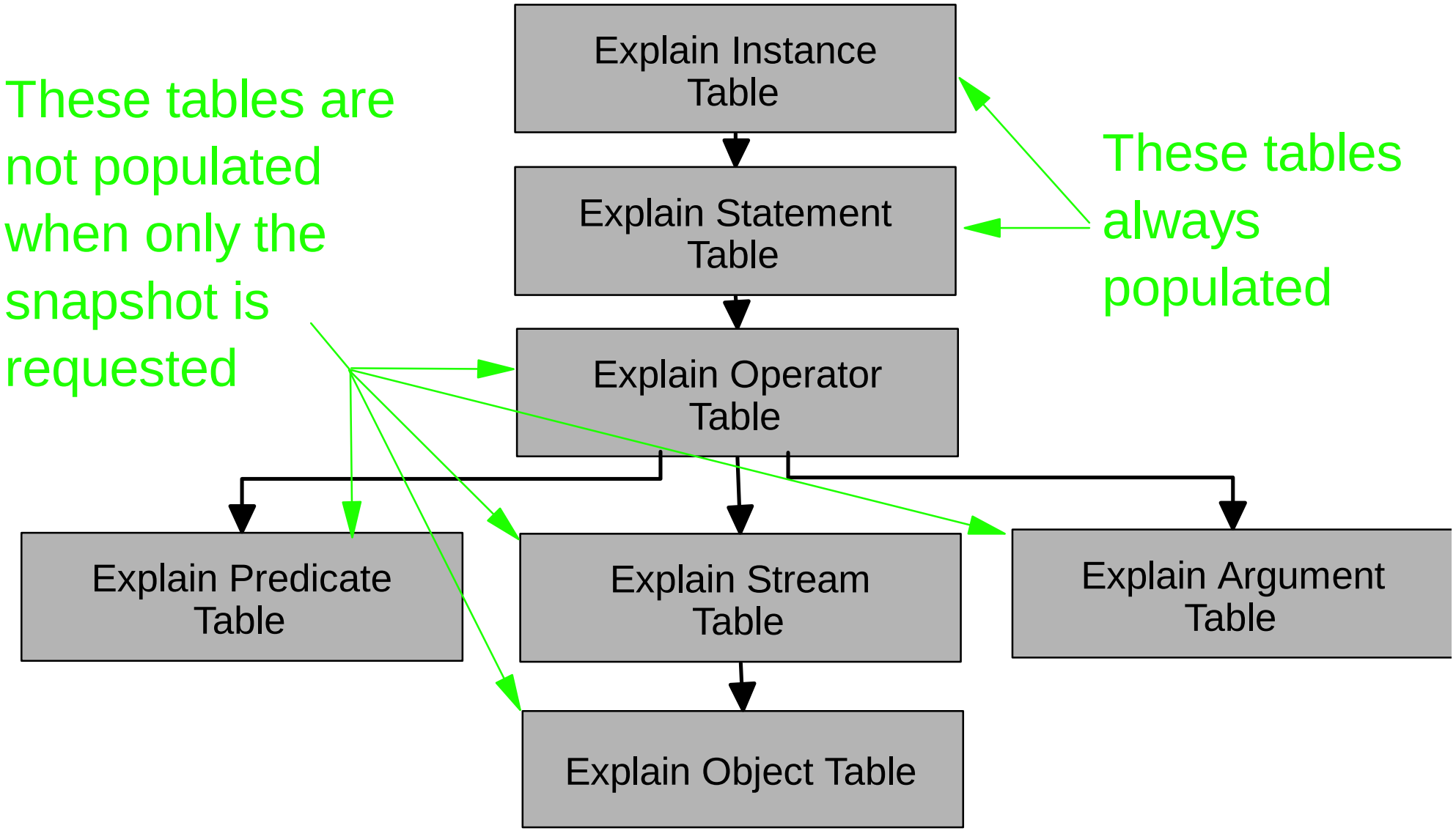
Why is Explain Important?

- Offers clues as to why the optimizer has made particular decisions
- Allows DBA to maintain a history of problem query access plans during key transition periods
  - New index additions
  - Large data updates/additions
  - RUNSTATS changes
  - Release to Release migration
  - Significant DB or DBM configuration changes
- Problem determination is easier, and often faster with a reference plan to compare against

How do I use Explain?

- Need to create the Explain Tables first
  - Visual Explain creates the Explain Tables automatically
  - You can create the Explain Tables manually using EXPLAIN.DDL, found in the sqllib/misc directory
- There are 7 Explain Tables
  - Engine inserts details of selected plan into the explain tables
- Various details about the explain tables and tools can be found in:
  - V6 SQL Reference, Appendix K - Explain Tables, and Appendix L - Explain Registers
  - V6 Administration Guide, Chapter 22 - SQL Explain Facility, Appendix K - Explain Tables, and Appendix M - db2exfmt

Explain Table Relationships



How do I get data in the Explain Tables?

- Visual Explain allows you to enter queries and have them explained, as well as view previously explained queries

- Manually, in static or dynamic SQL, you can use several methods:
  - Explain Statement:
    - Prepend explain plan with snapshot for to SQL statement
    - Prepend explain plan for to SQL statement
  - Explain special registers (for dynamic SQL only)
    - set explain mode to yes, no, or explain
    - set explain snapshot set to yes, no, or explain
  - Explain bind options (for static and/or dynamic SQL)
    - explain set to yes, no, or all
    - explsnap set to yes, no, or all

My Explain tables are getting big. Why?

- The engine only inserts into the explain tables
- The user or DBA must decide when plans no longer have value and clean up the explain tables.
- There is a remarks column in the explain instance table that can be manually updated to provide some details about when the explain instance was gathered
- There are querytag, and queryno columns in the explain statement table that can be set in the explain statement, or updated manually to provide additional details about the particular statement

What else can I do with the Explain information?

- Run custom queries to mine information about your explain plans
  - Search for sorts, or group bys on sets of columns and base tables that occur frequently, and could be beneficial as an index or AST
  - Search for expensive operations, large or spilling sorts, high buffer usage, high TQ usage, etc...
  - Search for expensive plans to further examine for database optimizations
  - Search for common predicates that could form potential start/stop keys for an index
  - Your imagination and SQL skills are your only limits!

Agenda

- The Explain Facility
  - What is it?
  - How to invoke it
  - Viewing options: Visual Explain and db2exfmt
  - What about db2expln and dynexpln?
- The Query Access Plan
  - Optimized SQL
  - Plan operators, arguments and properties
  - Predicate application
  - Execution flow (serial and parallel)
- Performance analysis tips

The Structure of Explain Information

- DB and DBM overview
  - Software release level
  - Basic Database configuration parameters
- Original SQL Statement text
  - The SQL statement as it was presented to the DB2 engine
- "Optimized" SQL Statement text
  - SQL-like representation of the query after it has been rewritten, views merged, constraints and triggers added
- Access Plan
  - An overview graph of the query access plan
  - Details of the LOw LEvel Plan OPerators (LOLEPOPs)

Understanding db2exfmt output

- Explain level, version and build level, application

```
DB2 Universal Database Version 6, 5622-044 (c)
Copyright IBM Corp. 1995, 1999
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool


******************** EXPLAIN INSTANCE *****************

DB2_VERSION:          06.01.0
BUILD LEVEL:          db2_v6:n990611

SOURCE_NAME:          SQLC39A3
SOURCE_SCHEMA:        NULLID
EXPLAIN_TIME:         1999-06-14-02.32.56.165498
EXPLAIN_REQUESTER:    HAIDER
```

Database context information

- Database and database manager configuration parameters considered by the optimizer

```
Database Context:
----------------
    Parallelism:              Inter-Partition Parallelism
    CPU Speed:                1.255649e-06
    Comm Speed:               2
    Buffer Pool size:         95000
    Sort Heap size:           22000
    Database Heap size:       4800
    Lock List size:           3500
    Maximum Lock List:        6
    Average Applications:     1
    Locks Available:          23730
```

Package information

- Optimization level, static or dynamic SQL, isolation level

```
Package Context:
---------------
    SQL Type:                Dynamic
    Optimization Level:      7
    Blocking:                Block All Cursors
    Isolation Level:         Repeatable Read
------------ STATEMENT 1  SECTION 201 -----------
    QUERYNO:           1
    QUERYTAG:          CLP
    Statement Type:    Select
    Updatable:         No
    Deletable:         No
    Query Degree:      1
```

Original and optimized SQL

- Example: TPC-R Q1

```
Original Statement:
------------------
select l_returnflag, l_linestatus, sum(l_quantity) as
sum_qty, sum(l_extendedprice) as sum_base_price,
sum(l_extendedprice * (1 - l_discount)) as
sum_disc_price, sum(l_extendedprice * (1 - l_discount) *
(1 + l_tax)) as sum_charge, avg(l_quantity) as avg_qty,
avg(l_extendedprice) as avg_price, avg(l_discount) as
avg_disc, count(*) as count_order
from tpcd.lineitem
where l_shipdate <= date ('1998-12-01') - 90 day
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus
```

Optimized SQL

```
Optimized Statement:
--------------------
SELECT Q3.$C7 AS "L_RETURNFLAG",
 Q3.$C6 AS "L_LINESTATUS", Q3.$C5 AS "SUM_QTY",
 Q3.$C4 AS "SUM_BASE_PRICE", Q3.$C3 AS "SUM_DISC_PRICE",
Q3.$C2 AS "SUM_CHARGE",(Q3.$C5 / Q3.$C0) AS "AVG_QTY",
(Q3.$C4 /Q3.$C0) AS "AVG_PRICE", (Q3.$C1 / Q3.$C0) AS
"AVG_DISC", INTEGER(Q3.$C0) AS "COUNT_ORDER"
FROM
   (SELECT SUM(Q2.$C2), SUM(Q2.$C3), SUM(Q2.$C4),
SUM(Q2.$C5),SUM(Q2.$C6),SUM(Q2.$C7), Q2.$C0, Q2.$C1
FROM
    (SELECT Q1.L_LINESTATUS, Q1.L_RETURNFLAG, Q1.COUNT,
Q1.S5, Q1.S4, Q1.S3, Q1.S2, Q1.S1
     FROM TPCD.L_SUMMARY AS Q1
      WHERE (Q1.L_SHIPDATE <= '09/02/1998')) AS Q2
   GROUP BY Q2.$C1, Q2.$C0) AS Q3
ORDER BY Q3.$C7, Q3.$C6
```

Internal column naming convention

- Internal name represents position in select list

```
Optimized Statement:
--------------------
SELECT Q3.$C7 AS "L_RETURNFLAG", ...
FROM
   (SELECT SUM(Q2.$C2), SUM(Q2.$C3), SUM(Q2.$C4),
SUM(Q2.$C5),SUM(Q2.$C6),SUM(Q2.$C7), Q2.$C0, Q2.$C1
FROM
    (SELECT Q1.L_LINESTATUS, Q1.L_RETURNFLAG, Q1.COUNT,
Q1.S5, Q1.S4, Q1.S3, Q1.S2, Q1.S1
     FROM TPCD.L_SUMMARY AS Q1
WHERE (Q1.L_SHIPDATE <= '09/02/1998')) AS Q2
   GROUP BY Q2.$C1, Q2.$C0) AS Q3
ORDER BY Q3.$C7, Q3.$C6
```

Optimized SQL

- Reflects effects of query rewrite optimizations
- For example - automatic redirection to summary tables

```
Original Statement:
-------------------
select ...
from tpcd.lineitem

Optimized Statement:
--------------------
SELECT ...
FROM
   (SELECT ...
    FROM
    (SELECT ...
     FROM TPCD.L_SUMMARY AS Q1
    ) AS Q2
   ) AS Q3
```

Optimized SQL

- Reflects effects of query rewrite optimizations
- For example - pre-computation of constant expressions

```
Original Statement:
-------------------
select ...
from tpcd.lineitem
where l_shipdate <= date ('1998-12-01') - 90 day

Optimized Statement:
--------------------
SELECT ...
FROM
   (SELECT ...
    FROM
    (SELECT ...
     FROM TPCD.L_SUMMARY AS Q1
     WHERE (Q1.L_SHIPDATE <= '09/02/1998')
    ) AS Q2
   ) AS Q3
```

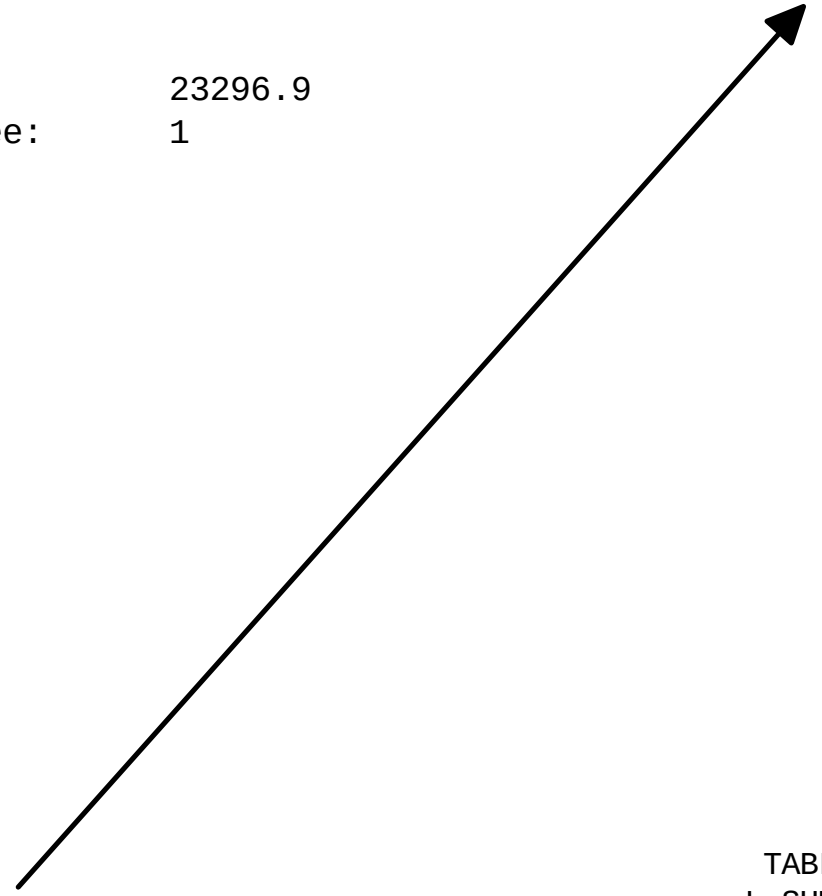
Optimized SQL

- Reflects effects of query rewrite optimizations
- For example - aggregation optimization

```
Original Statement:
-------------------
select ... sum(l_quantity) as sum_qty,
 avg(l_quantity) as avg_qty,
 count(*) as count_order
from ...

Optimized Statement:
--------------------
SELECT ... Q3.$C5 AS "SUM_QTY",
(Q3.$C5 / Q3.$C0) AS "AVG_QTY",
INTEGER(Q3.$C0) AS "COUNT_ORDER"
FROM ...
```

Access plan graph

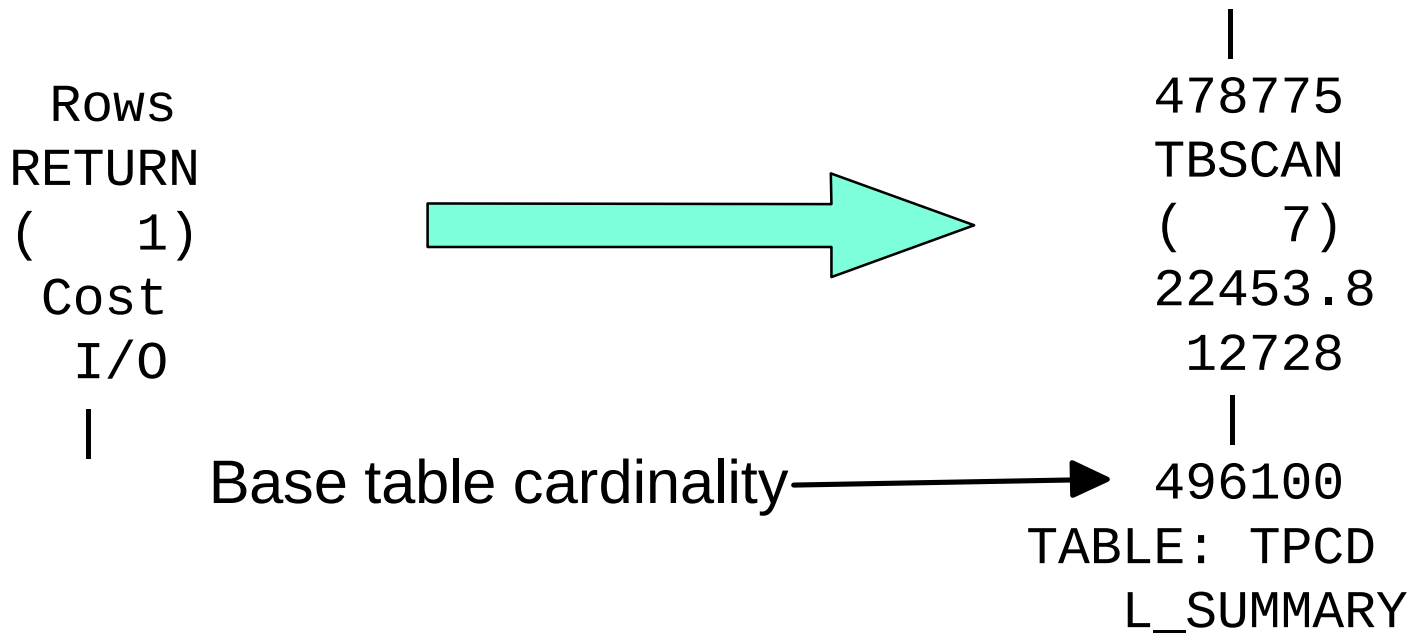
```
Access Plan:
-----------
     Total Cost:        23296.9
     Query Degree:      1

       Rows
      RETURN
      (   1)
       Cost
        I/O
        |
         6
      GRPBY
      (   2)
      23296.9
       12728
        |
        24
      MDTQ
      (   3)
      23296.9
       12728
        |
```



```
          6
        GRPBY
        (   4)
        23296.7
         12728
          |
          6
        TBSCAN
        (   5)
        23296.7
         12728
          |
          6
        SORT
        (   6)
        23296.7
         12728
          |
        478775
        TBSCAN
        (   7)
        22453.8
         12728
          |
        496100
     TABLE: TPCD
    L_SUMMARY
```

Access plan graph

- Data flow diagram comprised of plan operators
- Detailed plan operator information follows plan graph
- Tip: -TIC option adds rows, total cost and I/O cost info to graph



Access plan operators

- Represent runtime operations
- Plan operators map to a set of runtime operators in access section
- Base access methods
  - TBSCAN, IXSCAN, FETCH
- Joins
  - Nested loop join: NLJOIN
  - Merge scan join: MSJOIN
  - Hash join: HSJOIN
- Aggregation: GRPBY
  - See optimized SQL for actual aggregation operation e.g. SUM, MIN/MAX etc.
- Temping (TEMP) and sorting (SORT)

Access plan operators

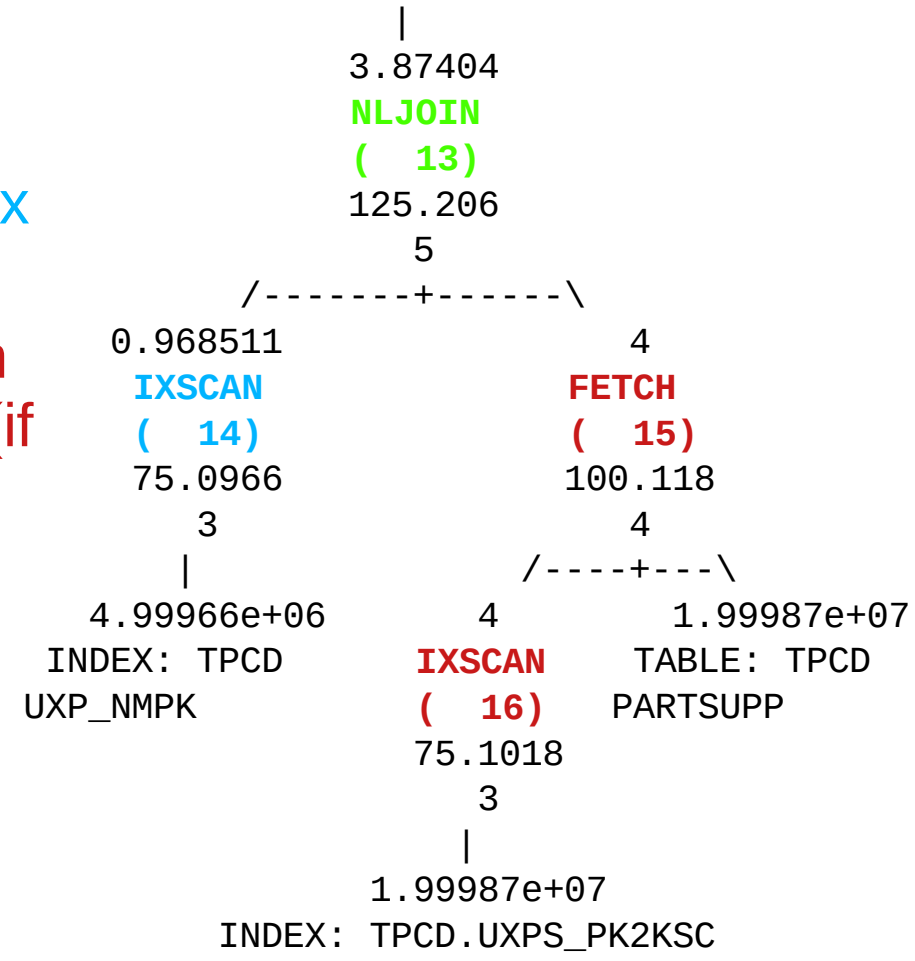
- Specialized operations
  - Index ANDing (IXA), dynamic bit map indexing
  - Index ORing and list prefetch (RIDSCN)
  - Star join uses dynamic bit map indexing (IXA)
  - Table queues (TQ)
    - Broadcast (BTQ)
    - Directed (DTQ)
    - Merging option (MDTQ, MBTQ)
    - Local table queue for SMP intra-partition parallelism (LTQ)

Access plan examples

- Nested loop join of PART and PARTSUPP tables

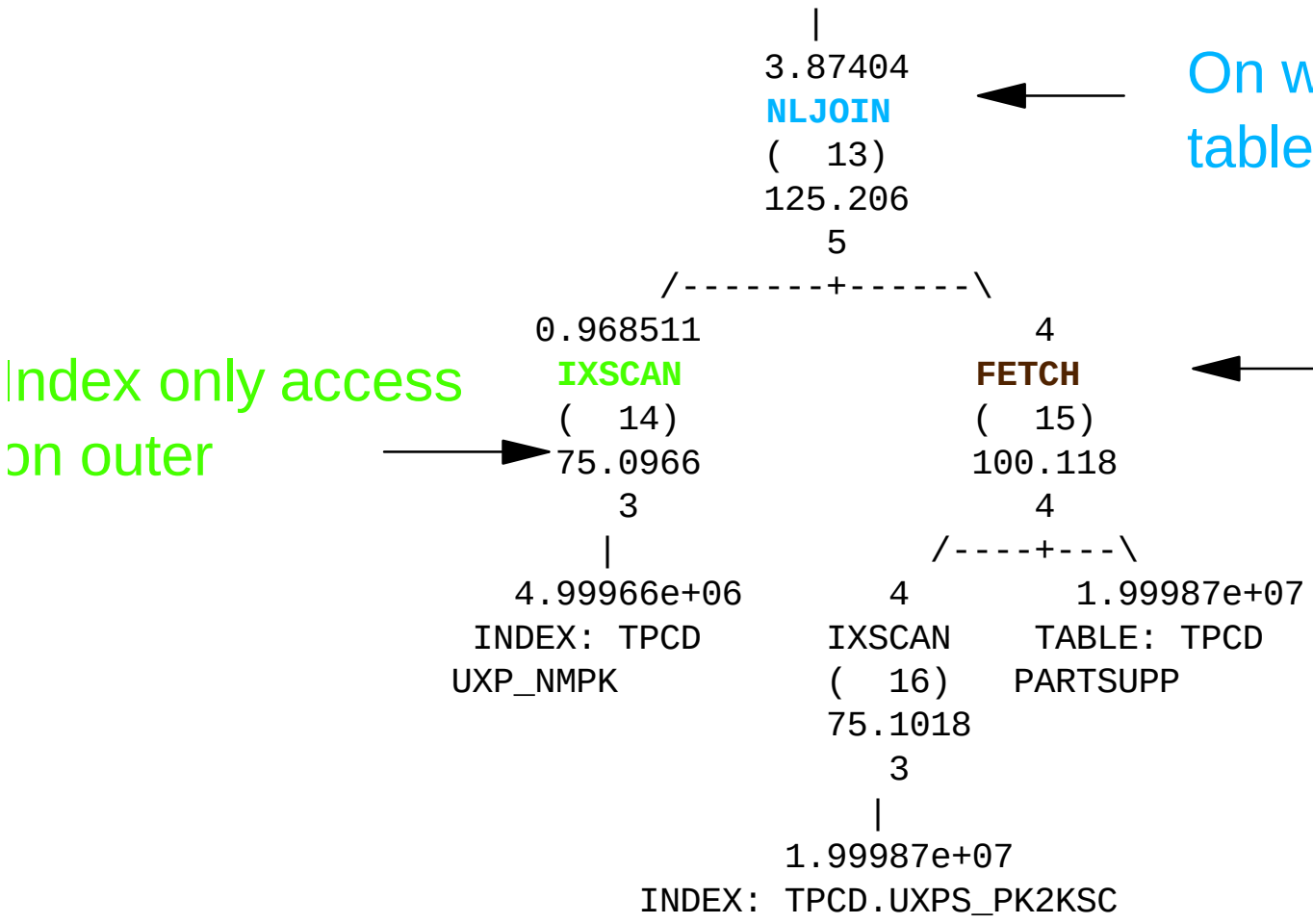
Execution flow is bottom up from left to right.

1. IXSCAN (14)retrieves row from index
2. Row is passed to NLJN (13)
3. NLJN accesses inner table based on join predicates and local predicates (if any) (FETCH(15) and ISCAN(16))
4. Each joined row is returned from NLJN to next operator
5. Execution continues until outer stream is exhausted



Access plan examples

- Nested loop join of PART and PARTSUPP tables



Plan operator details

- Use plan operator details to understand how query execute

```
13) NLJOIN: (Nested Loop Join)
        Cumulative Total Cost:          125.206
        Cumulative CPU Cost:            164264
        Cumulative I/O Cost:            5
        Cumulative Re-Total Cost:       0.062461
        Cumulative Re-CPU Cost:         49744
        Cumulative Re-I/O Cost:         0
        Cumulative First Row Cost:      125.204
        Estimated Bufferpool Buffers:   6

        Arguments:
        ---------
        EARLYOUT: (Early Out flag)
             FALSE
        FETCHMAX: (Override for FETCH MAXPAGES)
             IGNORE
        ISCANMAX: (Override for ISCAN MAXPAGES)
             IGNORE
```

Plan costs
(this operator and its subplans)

Operator arguments

Plan operator costs

- Total cost in units of timerons
  - Based on milliseconds
    - Not elapsed time in serial environment
    - Elapsed time-based in parallel environment
- Cost model is based on resource consumption
  - Total CPU and I/O resources consumed
- Communication costs are considered in parallel environment
- Elapsed time could be different because of parallel I/O and overlap between CPU and I/O operations in a serial environment
- Plan costs are cumulative
  - In general, each plan operator adds cost to the plan

Operator cost components

```
13) NLJOIN: (Nested Loop Join)
   Cumulative Total Cost:          125.206
   Cumulative CPU Cost:            164264
   Cumulative I/O Cost:            5

   Cumulative Re-Total Cost:       0.062461
   Cumulative Re-CPU Cost:         49744
   Cumulative Re-I/O Cost:         0

   Cumulative First Row Cost: 125.204

   Estimated Bufferpool Buffers: 6
```

Cumulative costs

Cost to re-execute subplan

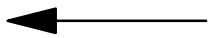
Total cost to return first row

Bufferpool pages required by this operator

Operator arguments

- Provide details on how operator executes
- See the Administration Guide, Appendix K - Explain Tables and Definitions, for complete description

```
13) NLJOIN: (Nested Loop Join)
 Arguments:
    ---------
    EARLYOUT: (Early Out flag)
        FALSE

    FETCHMAX: (Override for FETCH MAXPAGES)
        IGNORE

    ISCANMAX: (Override for ISCAN MAXPAGES)
        IGNORE
```



Get next outer after finding first match on inner. Guaranteed one match on inner.

Maximum pages to prefetch for FETCH and ISCAN.

NLJOIN can override original settings if an 'ordered' NLJOIN.

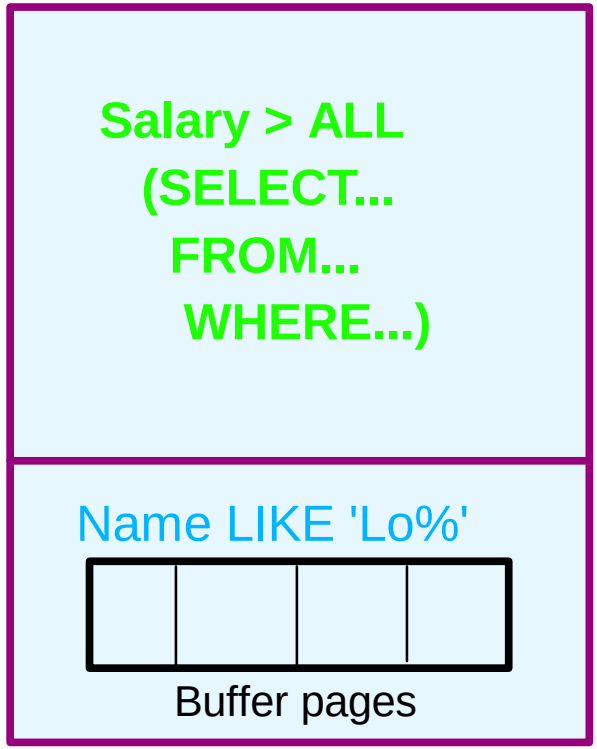
Operator predicates

- Sargable (Search ARGument) predicates
  - Applied by data manager or index manager without copying data from data or index page
- Residual predicates
  - Data copied from page to buffer, predicate applied by relational data service (RDS) runtime layer
- Start/stop key predicates
  - Applied by index scan
- Subquery predicates
  - A subplan must be executed
  - Results may be temped
  - Could be correlated
  - Applied as residual predicates

Hierarchy of Predicate Application

Residual Predicates

Search Arguments (SARGs)

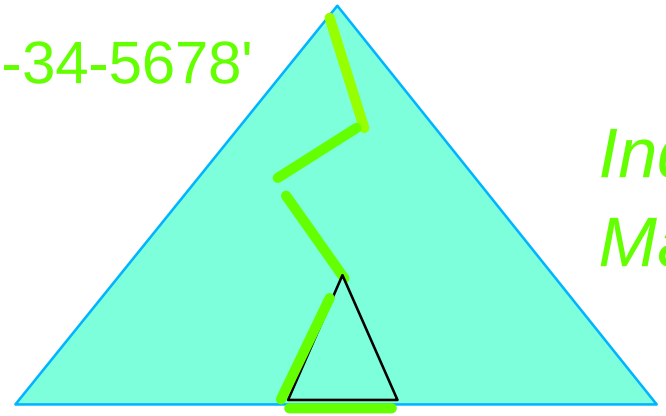


RDS

Data Manager

SSN= '012-34-5678'

Index Start/Stop Conditions (keycols)



Index Manager

Operator predicates

- NLJOIN predicate example

```
13) NLJOIN: (Nested Loop Join)

 Predicates:
 ----------
 16) Predicate used in Join
     Relational Operator:      Equal (=)
     Subquery Input Required:  No
     Filter Factor:            5.00034e-08

     Predicate Text:
     --------------
    (Q1.PS_PARTKEY = Q2.P_PARTKEY)
```

Predicate selectivity estimate

Predicate text based on optimized SQL

Operator input and output streams

- NLJOIN example

```
13) NLJOIN: (Nested Loop Join)
Input Streams:
-------------
    5) From Operator #14
        Estimated number of rows:  0.968511
        Partition Map ID:      1
        Partitioning:          (MULT )
                                   Multiple Partitions
        Number of columns:     3
        Subquery predicate ID:     Not Applicable

        Column Names:
        ------------
        +$RID$+P_PARTKEY+P_NAME

        Partition Column Names:
        ----------------------
        +1: PS_PARTKEY
```

Stream cardinality

Partitioning information

Stream information

Operator input and output streams

- NLJOIN has 2 input streams, 1 output stream

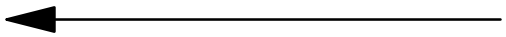
```
13) NLJOIN: (Nested Loop Join)
Input Streams:
-------------
    9) From Operator #15
       Estimated number of rows:   4
       Partition Map ID:           1
       Partitioning:               (MULT )
                                   Multiple Partitions
       Number of columns:          4
       Subquery predicate ID:      Not Applicable

       Column Names:
       ------------
       +PS_PARTKEY(A)+PS_SUPPKEY(A)+$RID$+PS_AVAILQTY

        Partition Column Names:
        ----------------------
        +1: PS_PARTKEY
```

Inner cardinality is per outer

Insight from operator details

- Why was FETCH necessary on inner of NLJOIN(13) ?

```
15) FETCH : (Fetch)
    Arguments:
    ---------
    ...
    Input Streams:
    -------------
        7) From Operator #16
                Column Names:
                ------------
                +PS_PARTKEY(A)+PS_SUPPKEY(A)+$RID$

        8) From Object TPCD.PARTSUPP
                Column Names:
                ------------
                +PS_AVAILQTY
```



Column in stream is not in index

Insight from operator details

- How is NLJOIN join predicate applied ?

```
.6) IXSCAN: (Index Scan)
        Predicates:
        ----------
        16) Start Key Predicate
            Relational Operator:        Equal (=)
            Subquery Input Required:    No
            Filter Factor:              5.00034e-08

            Predicate Text:
            --------------
            (Q1.PS_PARTKEY = Q2.P_PARTKEY)

        16) Stop Key Predicate
            Relational Operator:        Equal (=)
            Subquery Input Required:    No
            Filter Factor:              5.00034e-08

            Predicate Text:
            --------------
            (Q1.PS_PARTKEY = Q2.P_PARTKEY)
```

Applied as start/stop key predicate on IXSCAN. Estimate 4 matching inner rows per outer.

Operator Details - Did that SORT spill?

- The details for a SORT will indicate if the SORT spilled
- The I/Os indicate that there was I/O, and thus spilling associated with the SORT.
- The Estimated Bufferpool Buffers associated with the TBSCAN above the SORT indicate how many pages of spilling there was

```
        |
  3.65665e+07
    TBSCAN
    (  15)
  6.87408e+06
  1.45951e+06
        |
  3.65665e+07
    SORT
    (  16)
  6.14826e+06
  1.30119e+06
        |
  3.65665e+07
    TBSCAN
    (  17)
  2.00653e+06
  1.14286e+06
        |
  3.74999e+07
TABLE: TPCD
ORDERS
```

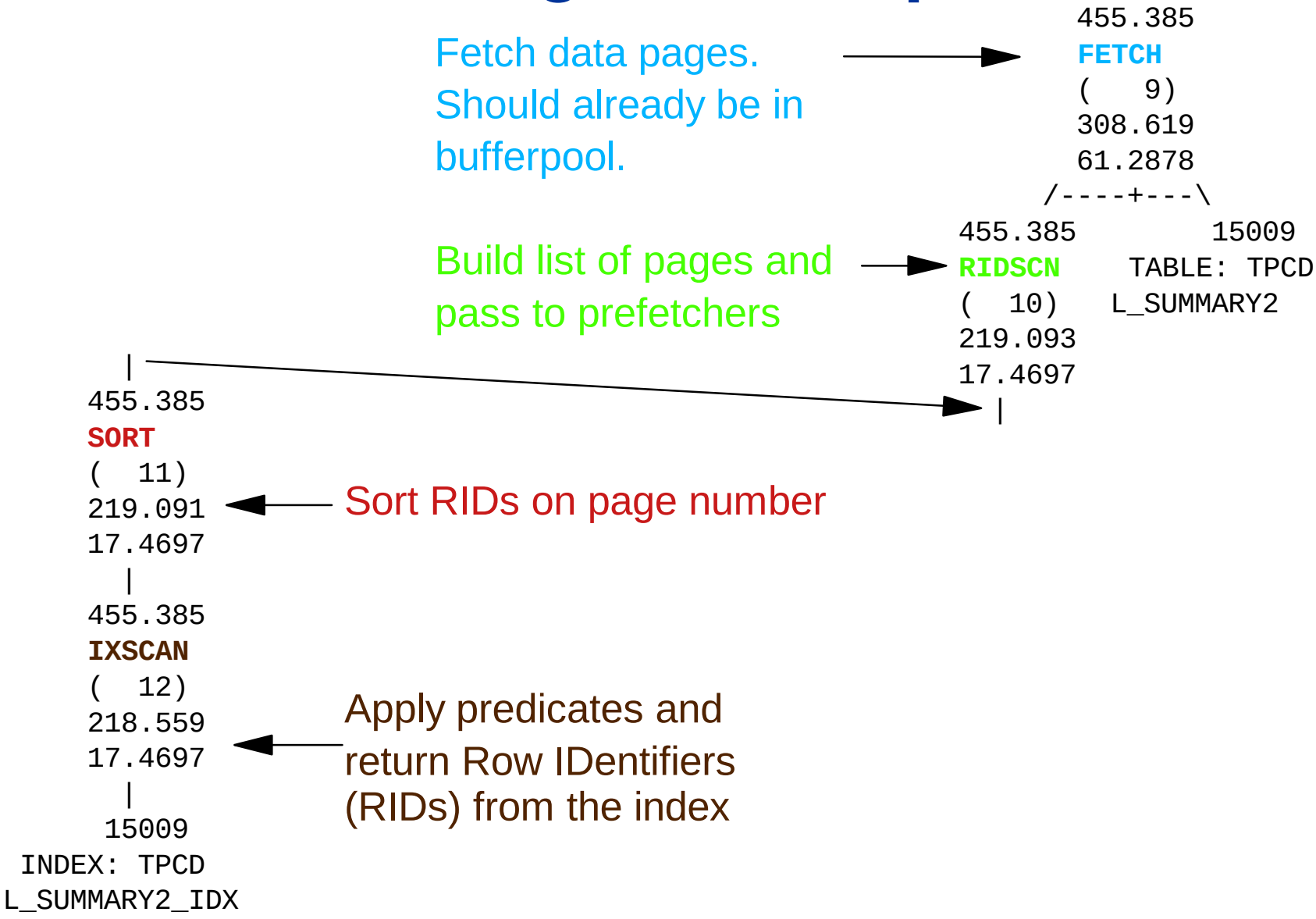
```
15) TBSCAN: (Table Scan)
                .
                .
                .
        Estimated Bufferpool Buffers:  163976
```

Operator Details - What about TQs

- Table Queue represents communication between nodes or subagents
- There are 4 key types of TQs:
  - Merging TQ (MDTQ, MBTQ)
  - Broadcast TQ (BTQ, MBTQ)
  - Directed TQ (DTQ, MDTQ)
  - Local TQ (LTQ, LMTQ)

```
3) TQ    : (Table Queue)
     Cumulative Total Cost:        9.59526e+06
     Cumulative CPU Cost:          5.34502e+11
     Cumulative I/O Cost:          5.36014e+06
     Cumulative Re-Total Cost:     9.33374e+06
     Cumulative Re-CPU Cost:       3.44133e+11
     Cumulative Re-I/O Cost:       0
     Cumulative First Row Cost:    9.59526e+06
     Estimated Bufferpool Buffers:      0

     Arguments:
     ---------
     LISTENER: (Listener Table Queue type)
          FALSE
     SORTKEY : (Sort Key column)
          1: L_RETURNFLAG(A)
     SORTKEY : (Sort Key column)
          2: L_LINESTATUS(A)
     TQMERGE : (Merging Table Queue flag)
          TRUE
     TQREAD  : (Table Queue Read type)
          READ AHEAD
     TQSEND  : (Table Queue Write type)
          DIRECTED
     UNIQUE  : (Uniqueness required flag)
          FALSE
```

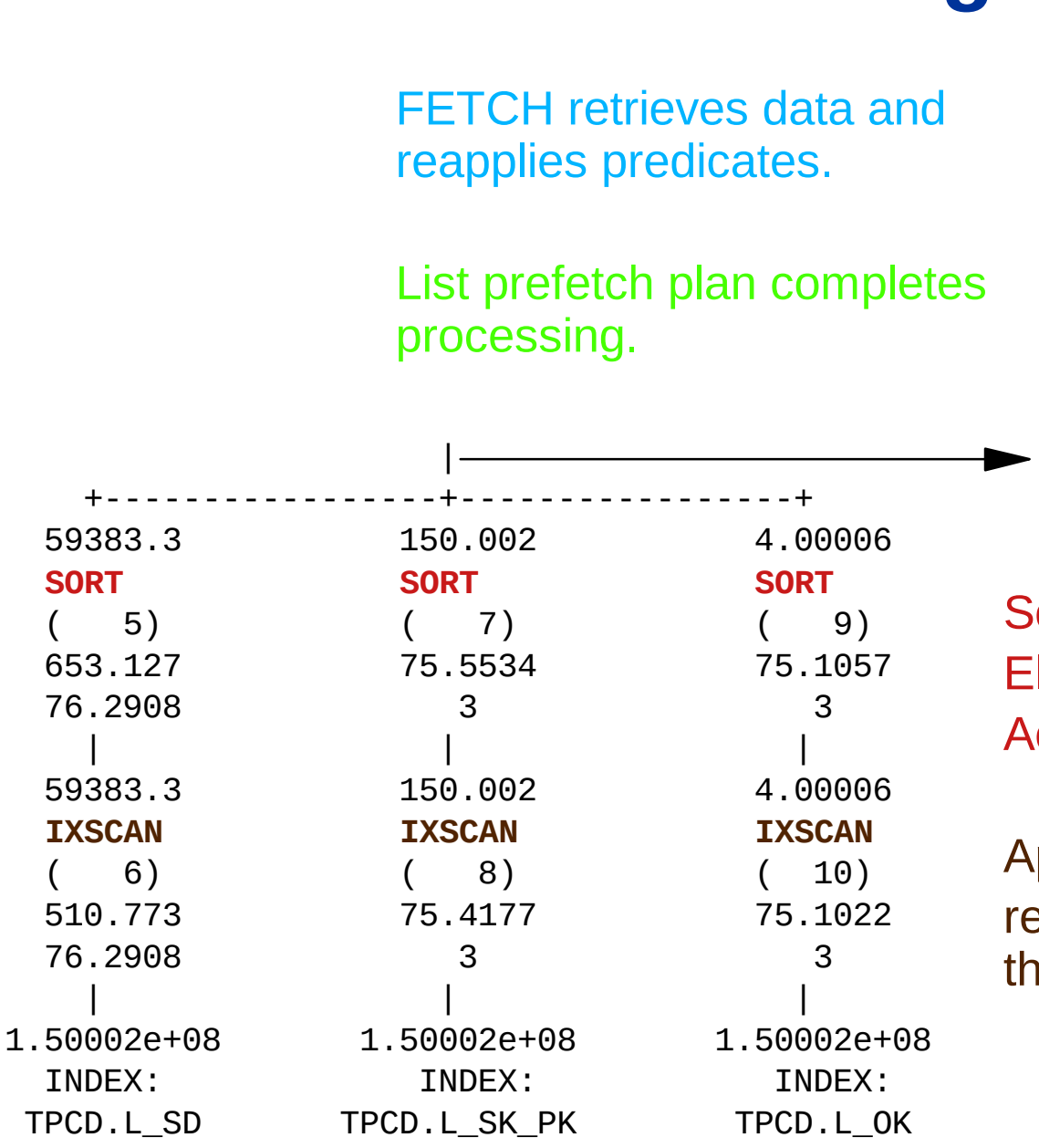
How do I recognize List prefetch?



What about Index ORing?

FETCH retrieves data and reapplies predicates.

List prefetch plan completes processing.



Sort RIDs on page number

Eliminate duplicate RIDs

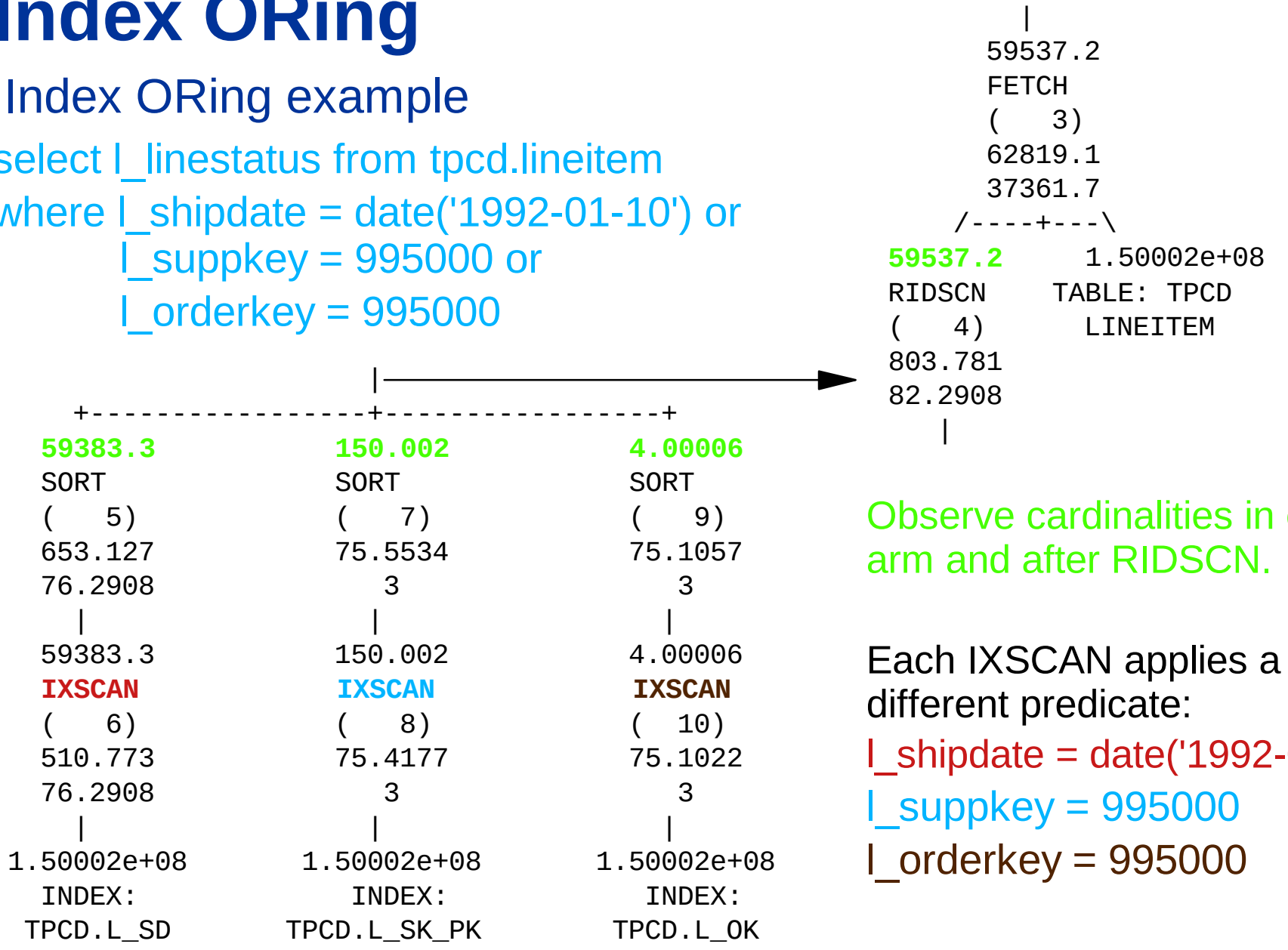
Actually the same sort for each arm

Apply predicates and return Row IDentifiers (RIDs) from the index

Index ORing

- Index ORing example

```
select l_linestatus from tpcd.lineitem
where l_shipdate = date('1992-01-10') or
      l_suppkey = 995000 or
      l_orderkey = 995000
```



Observe cardinalities in each arm and after RIDSCN.

Each IXSCAN applies a different predicate:

l_shipdate = date('1992-01-10')

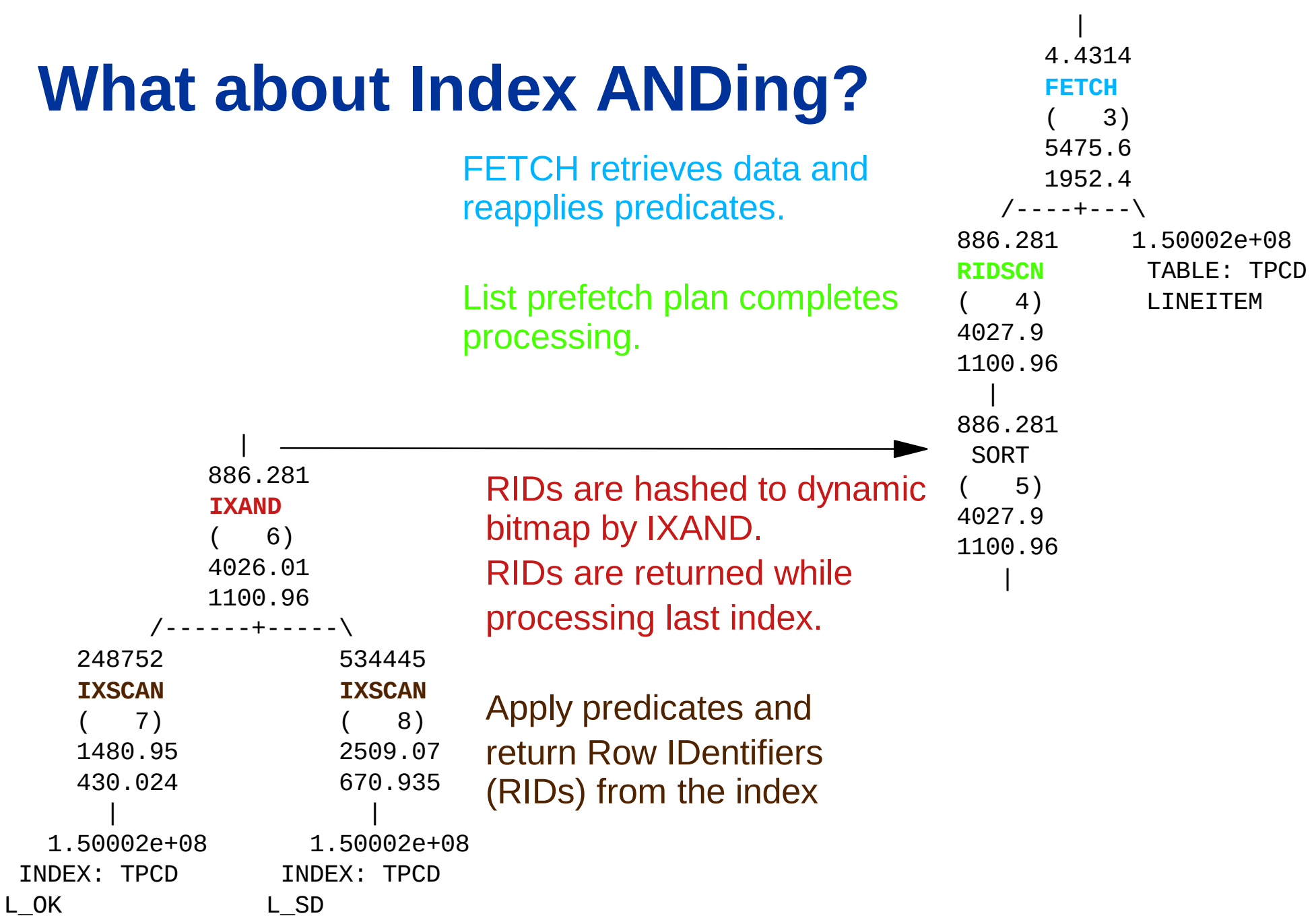
l_suppkey = 995000

l_orderkey = 995000

What about Index ANDing?

FETCH retrieves data and reapplies predicates.

List prefetch plan completes processing.



RIDs are hashed to dynamic bitmap by IXAND.

RIDs are returned while processing last index.

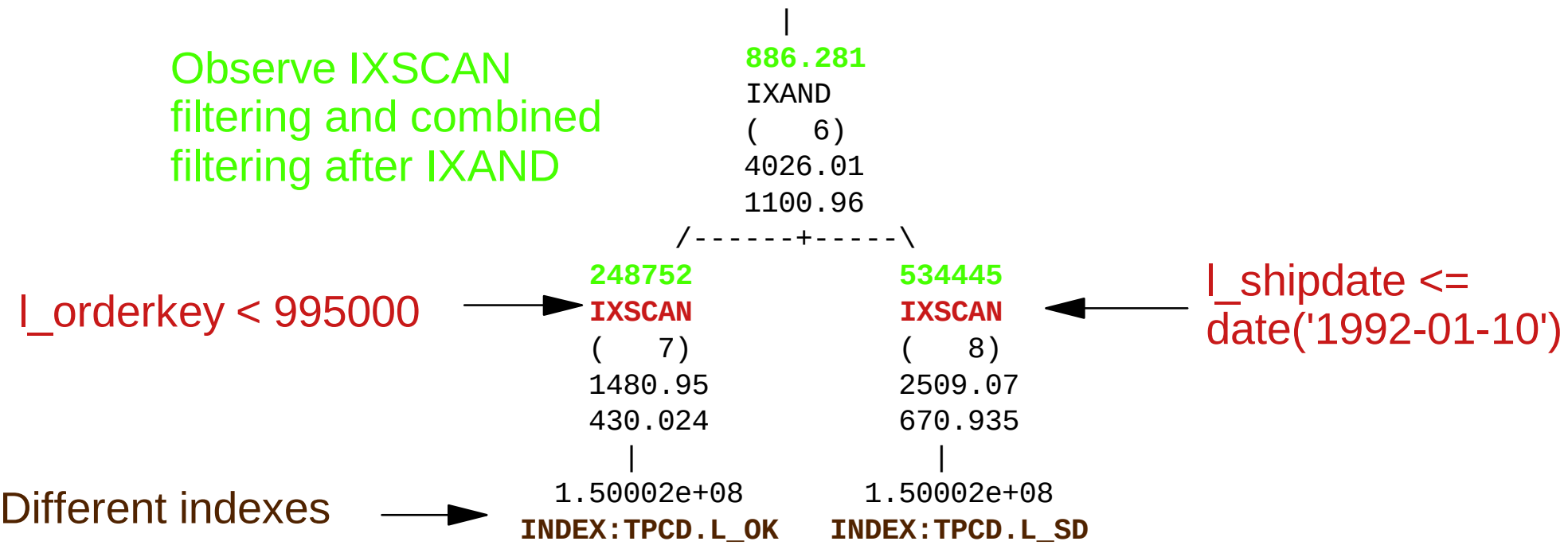
Apply predicates and return Row IDentifiers (RIDs) from the index

Index ANDing

- Index ANDing example

```
select l_linestatus from tpcd.lineitem
  where  l_shipdate <= date('1992-01-10') and
         l_suppkey > 995000 and
         l_orderkey < 995000;
```

Observe IXSCAN filtering and combined filtering after IXAND



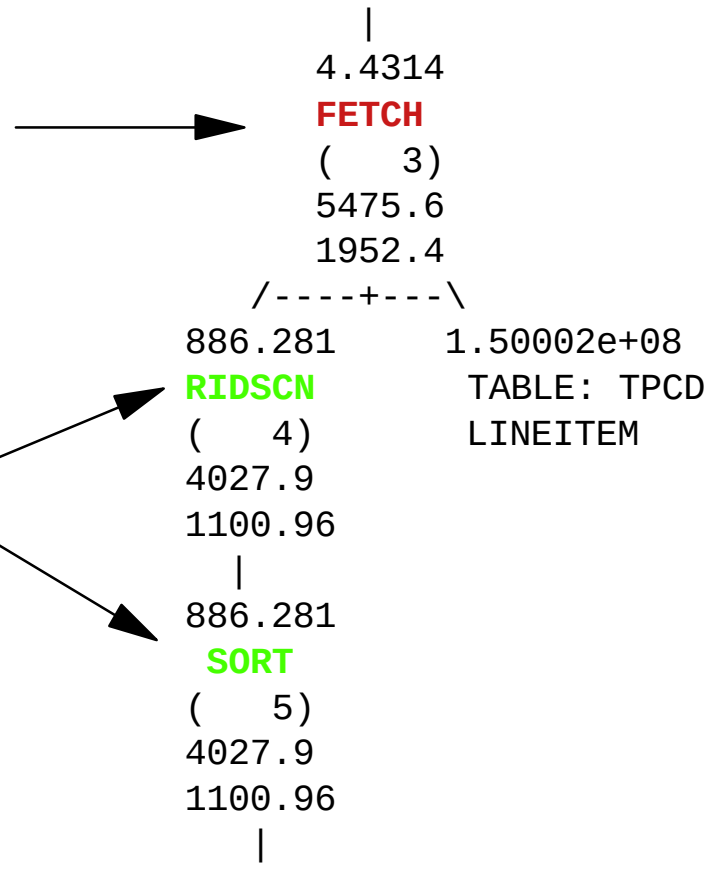
Index ANDing

- Index ANDing example

```
select l_linestatus from tpcd.lineitem
  where  l_shipdate <= date('1992-01-10') and
         l_suppkey > 995000 and
         l_orderkey < 995000;
```

IXSCAN predicates are reapplied by FETCH as well as any other eligible predicate.
(e.g. all 3 predicates applied)

List prefetch is used to retrieve data pages efficiently.



Star join

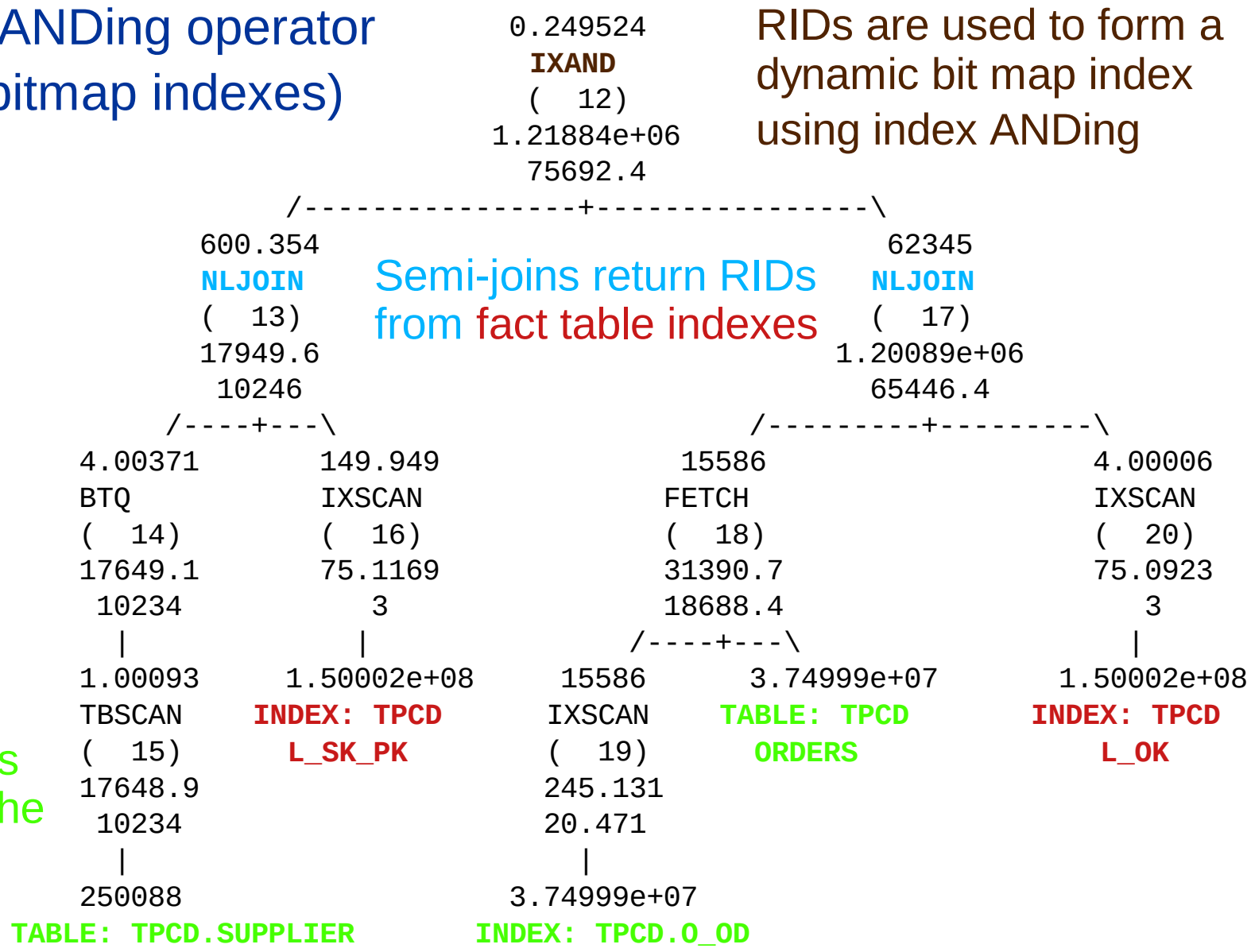
- Large fact table is filtered by multiple dimension tables or more complex dimensions called 'snowflakes'
- Schema is 'star' shaped with fact table in centre of star

```
select l_extendedprice, l_discount, l_quantity
from
tpcd.lineitem, tpcd.supplier, tpcd.orders, tpcd.part
where
l_suppkey = s_suppkey and
l_orderkey = o_orderkey and
l_partkey = p_partkey and
s_name = 'Supplier#000419963' and
o_orderdate = date('1996-12-31') and
p_type = 'ECONOMY ANODIZED STEEL'
```

Star join

- Uses index ANDing operator (dynamic bitmap indexes)

RIDs are used to form a dynamic bit map index using index ANDing



Semi-joins return RIDs from fact table indexes

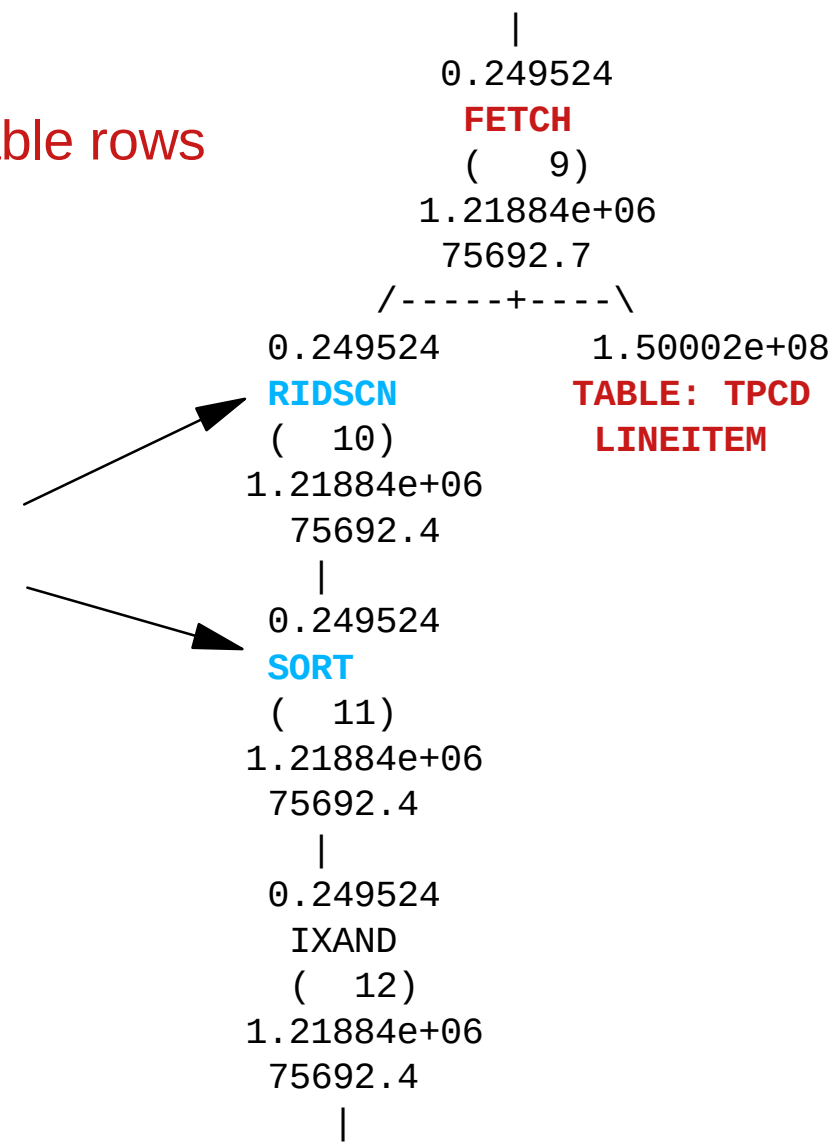
Dimension tables filter rows from the fact table

Star join

- Retrieving fact table rows

Fetch retrieves fact table rows

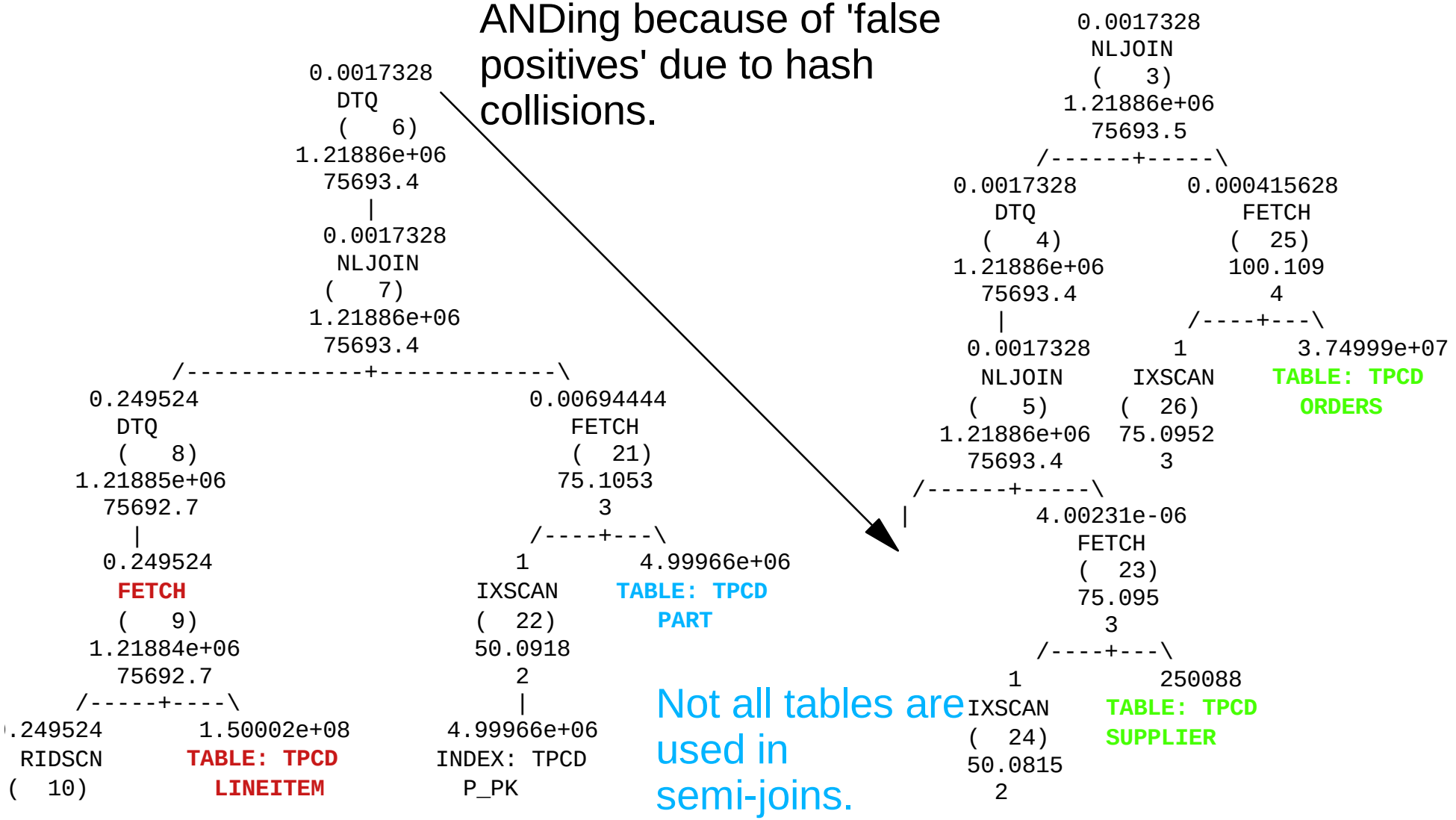
List prefetch operation is used to retrieve fact table data pages.



Star join

- 'Back' joins

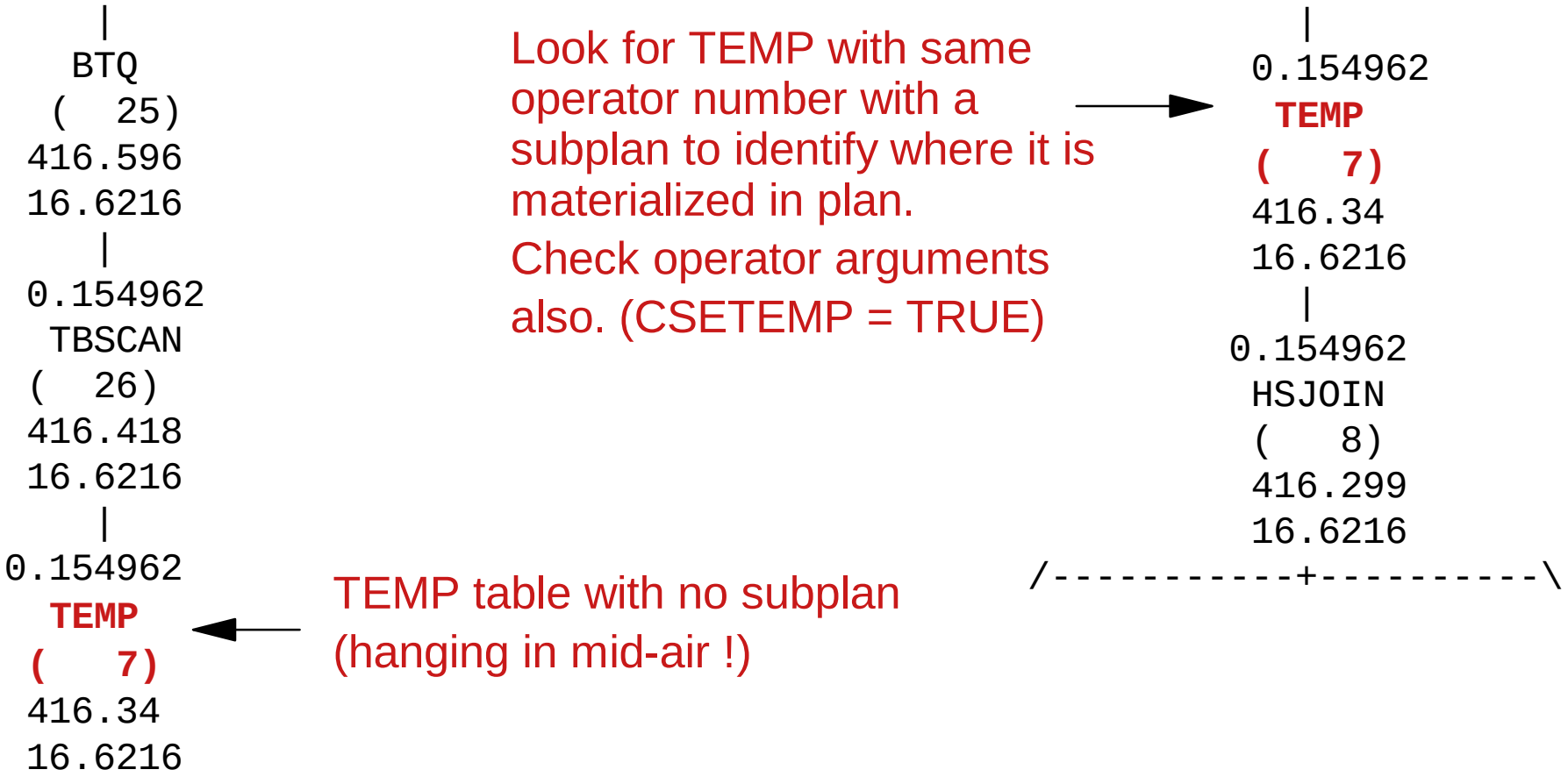
Dimension tables used in semi-joins must be rejoined with fact table rows after index ANDing because of 'false positives' due to hash collisions.



Not all tables are used in semi-joins.

Common sub-expression (CSE) TEMPs

- Created once and referenced multiple times



Agenda

- The Explain Facility
  - What is it?
  - How to invoke it
  - Viewing options: Visual Explain and db2exfmt
  - What about db2expln and dynexpln?
- The Query Access Plan
  - Optimized SQL
  - Plan operators, arguments and properties
  - Predicate application
  - Execution flow (serial and parallel)
- Performance analysis tips

Investigating Problem Queries

- db2look can be used to mimic, or simulate a database using update statistics
  - db2look extracts the DDL, and all of the relevant statistics that the optimizer requires to plan a query
  - Works well for EE or SMP systems,
  - May not work as well for EEE systems, since different numbers of nodes can affect the query plan chosen
    - Internally, we use a simulation tool to get around this problem
- System configuration is also important to the optimizer
  - plan selection is affected by both DBM and DB configuration parameters
  - not all parameters can be set in a way that is useful
    - Internally we have a way to "fake out" the optimizer

DBM Configuration parameters

- Optimizer is affected by the following Database Manager configuration parameters:
  - CPUSPEED
  - COMM_BANDWIDTH (for Intra-partition parallelism)
  - INTRA_PARALLEL

- Optimizer is affected by the following Database configuration parameters:
  - DFT_QUERYOPT
  - SORTHEAP
  - STMTHEAP
  - LOCKLIST
  - AVG_APPLS
  - MAXLOCKS
  - DBHEAP (Inter-partition parallelism)
  - DFT_DEGREE (Inter-partition parallelism)
  - BUFFPAGE (must account for multiple bufferpools)

What will I be looking for?

- missed index opportunities
- better join opportunities
- poor predicate selectivities due to insufficient statistics
- FETCH used because an index could use INCLUDE columns
- large TEMP or SORT
  - Did the SORT spill to disk? (I/O increased during SORT)
- Inspect TQ location
  - poor table partitioning decisions in DB design
  - TQing large amounts of data due to poor join predicates, or lack of local predicates
  - better collocation of tables

In Summary

- Explain provides insight into the optimizer's planning decisions
- There are two key tools to use to visualize the explain information:
  - Visual Explain provides a navigable interface to Explain
  - db2exfmt provides a text-based view of Explain
- More details about Explain can be found in:
  - V6 Administration Guide
    - Chapter 22 - SQL Explain Facility
    - Appendix K - Explain Tables and Definitions
    - Appendix M - db2exfmt - Explain Table Format Tool
  - V6 SQL Reference
    - Appendix K - Explain Tables and Definitions
    - Appendix L - Explain Register Values
  - Visual Explain's Online Help

In Summary - Other sessions of Interest

- Related talks include:
  - C8 - Using DB2 as a Tool to Select Indexes, by Danny Zilio
  - C11 (repeat of C2) - DB2 Optimizer's Secrets Revealed by Visual Explain, by Guy Lohman