Understanding Federated Query Performance DB2 Information Integrator V8.2

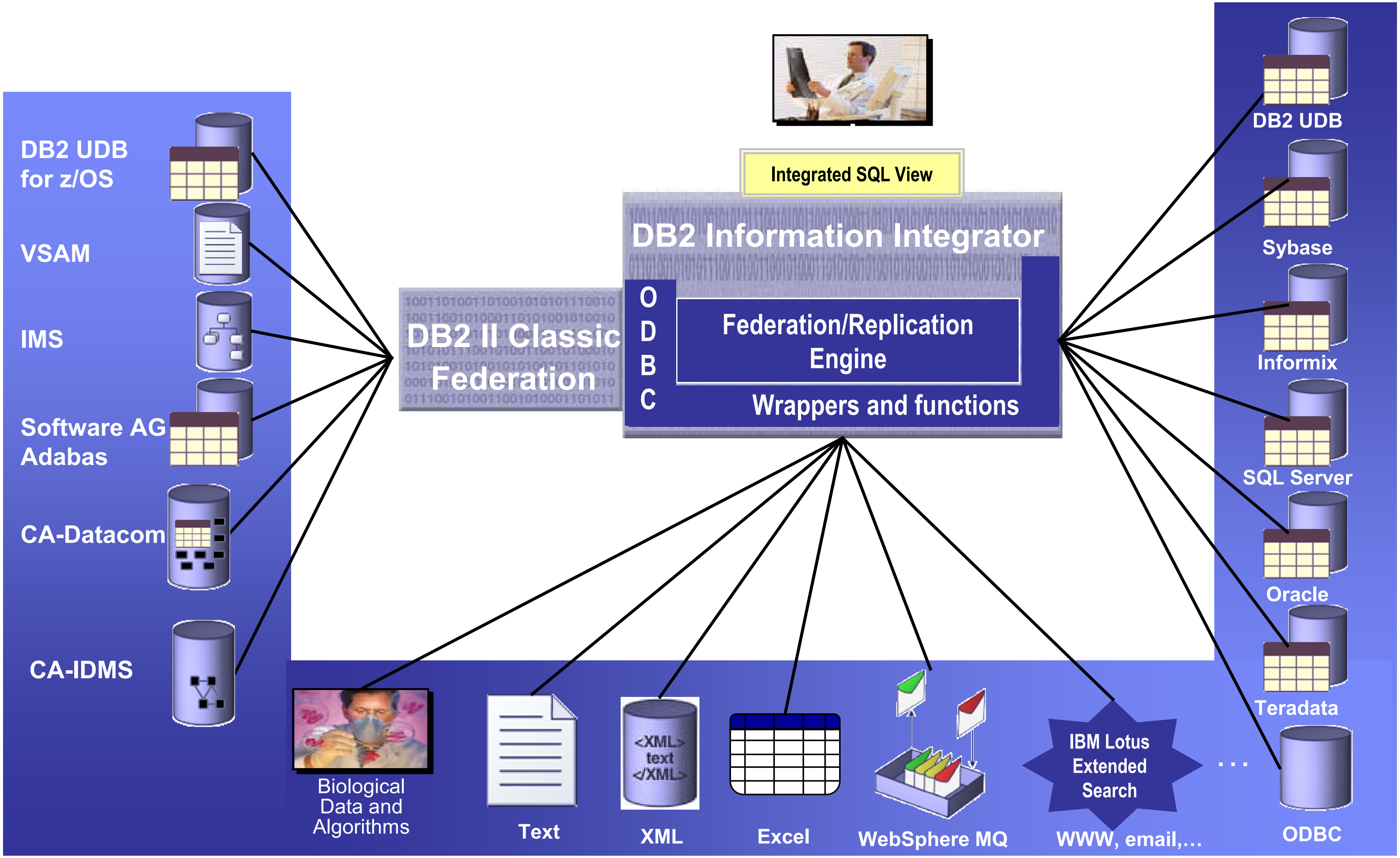
Susanne Englert
Senior Software Engineer
IBM Silicon Valley Lab
senglert@us.ibm.com

September 10, 2004

Agenda

- DB2 Information Integrator basics
  - Basic components, how it works, pushdown, what impacts performance?
- Federated queries against a single remote source
  - When is performance "good?" Reading an EXPLAIN. Diagnosing performance problems.
- Federated queries involving multiple sources
  - What does good performance mean? Useful comparisons
- Configuring DB2 II for best performance
  - Server options, type and function mappings. Statistics. MQTs.
- New features in II V8.2 that can improve performance
  - Parallel execution in SMP and MPP environments
  - Fenced wrapper, informational constraints, better monitoring
- Using DB2 II as a part of a solution
  - "Appropriate deployment" – when should you use it?
  - Know your workload! Complement DB2 II with caching/replication

Integrating Enterprise Data with DB2 Information Integrator

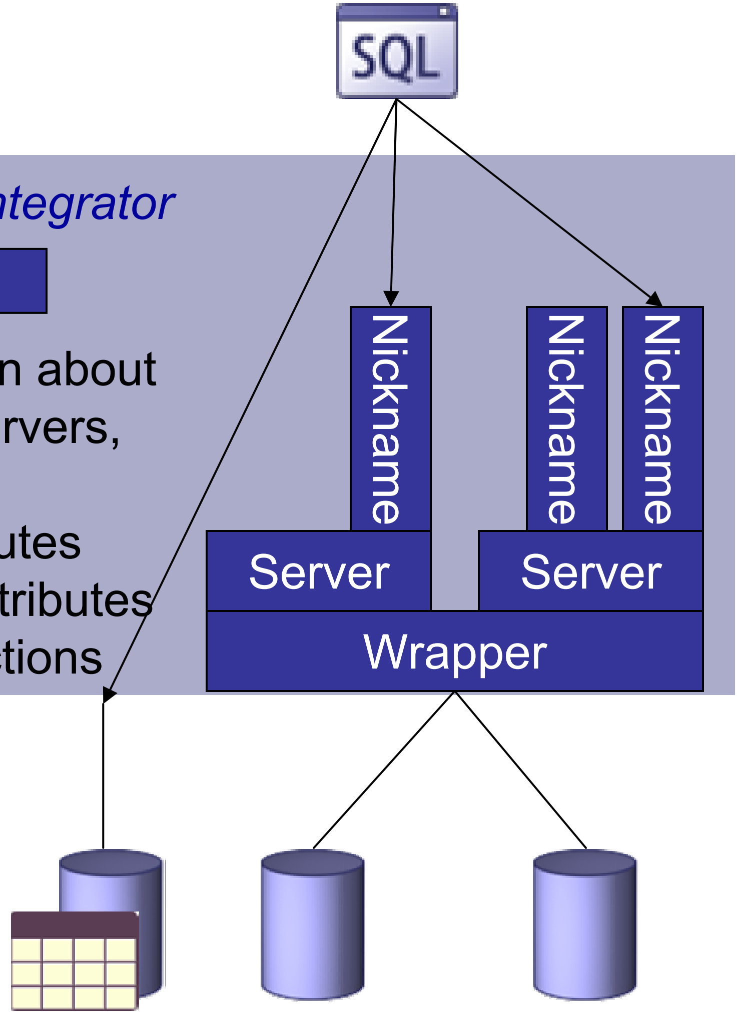


DB2 Information Integrator basic concepts I

DB2 Information Integrator

DB2 Catalog

- Stores information about
  - Wrappers,servers, nicknames
  - Server attributes
  - Nickname attributes
  - Remote functions



SQL

Nickname

Nickname

Nickname

Server

Server

Wrapper

Data at II instance:

- Global catalog
- *User data*
- MQTs

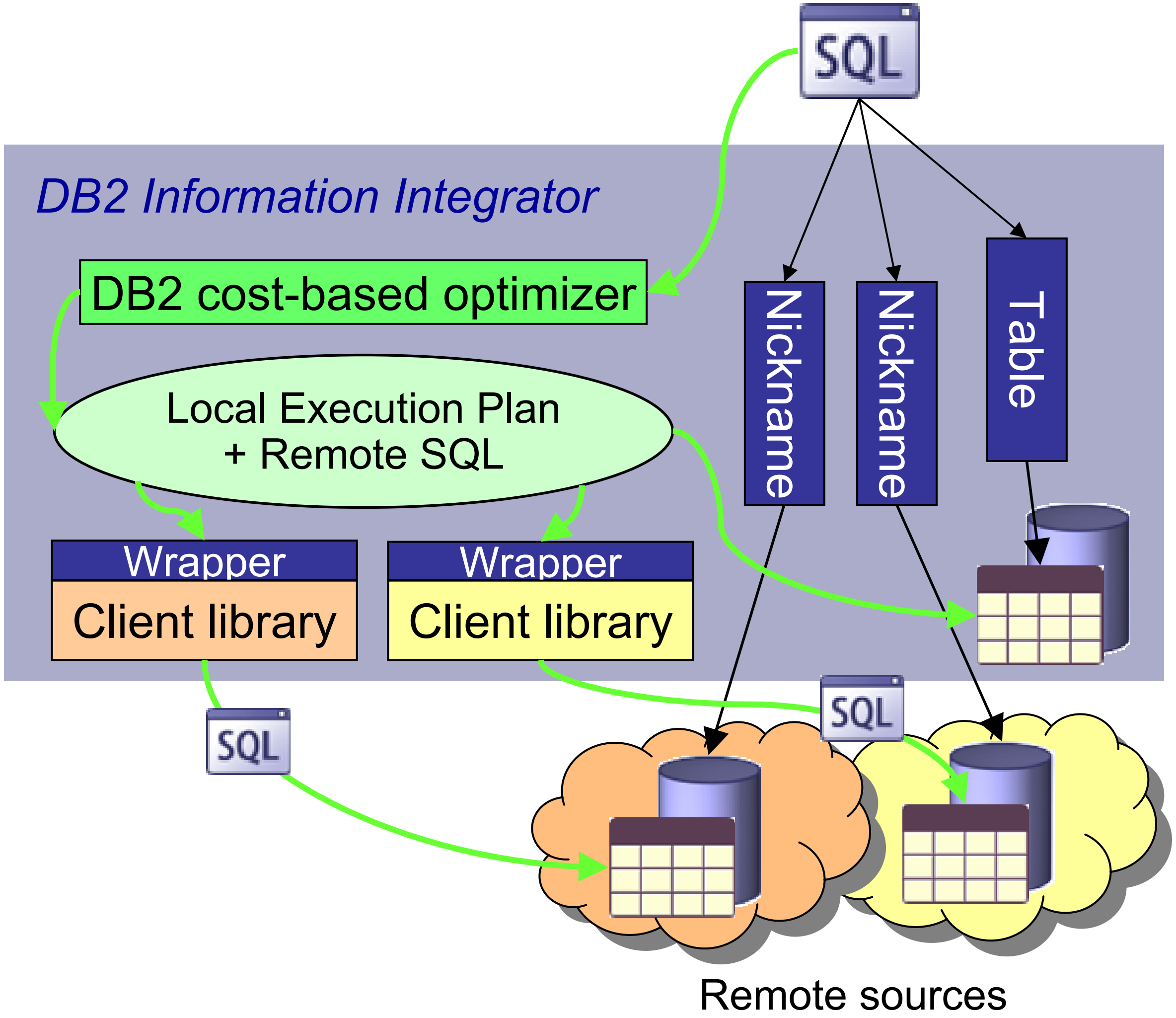
Wrapper: **a library allowing access to a particular class of data sources or protocols (Net8, DRDA, CTLIB...). Contains information about data source characteristics**

Server: **represents a specific data source**

Nickname: **a local alias to data on a remote server (mapped to rows and columns)**

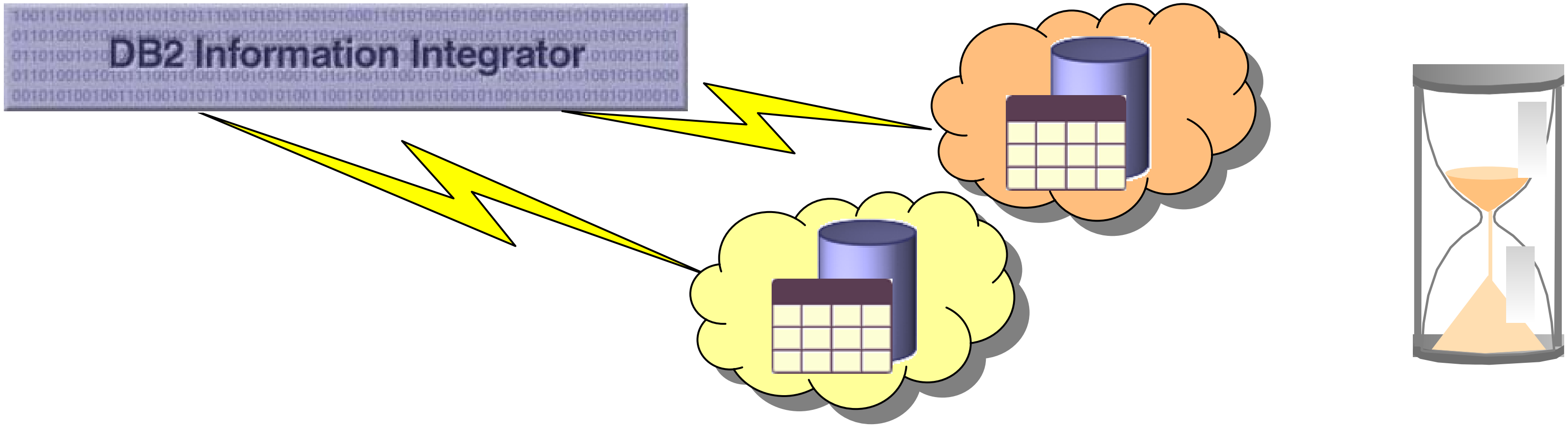
DB2 Information Integrator basic concepts II

- DB2 II is based on DB2 technology
- Nicknames look just like tables
  - to the application
  - inside the DB2 catalog
- Federated execution plans chosen by DB2 cost-based optimizer
- Optimizer decides how to distribute query work between DB2 II and remote sources. Cost-based pushdown of predicates and joins
- Query fragments executed remotely are sent via the native client library in the source's own dialect



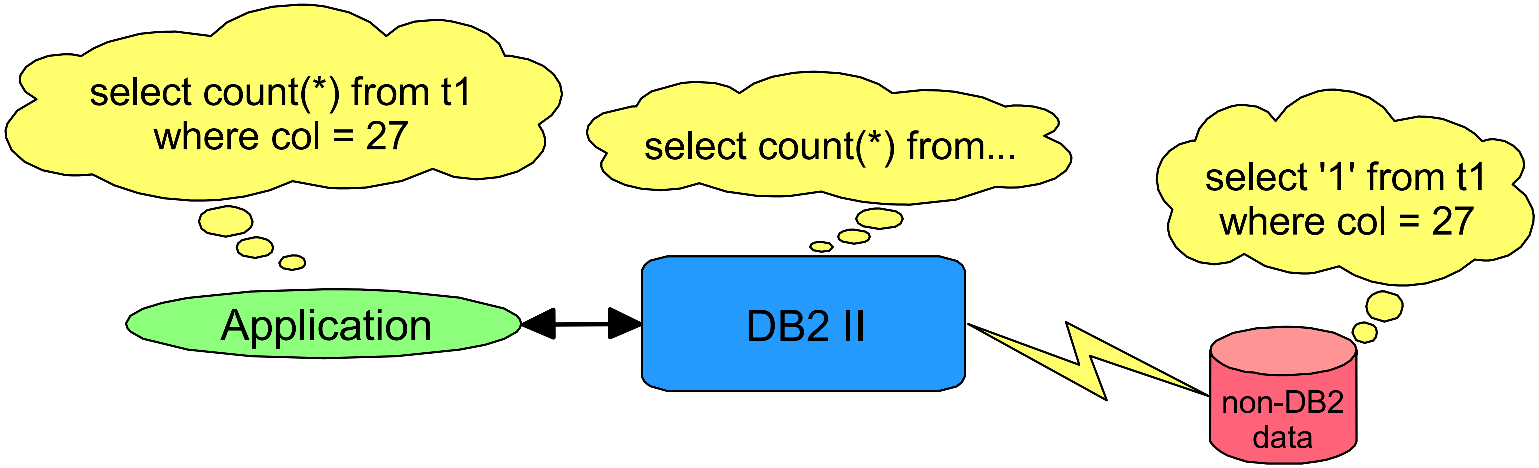
What factors affect DB2 II performance?

- Processing power, network bandwidth
- “Traffic” between remote sources and DB2 II server:
  - Number of interactions (roughly: requests to remote sources)
  - Amount of data moved from remote sources to DB2 II server
- Traffic depends on
  - Data placement among multiple sources – remember that data always moves from source to DB2 II server, never the other way around
  - Degree of processing "pushdown" to remote data source(s)
- Execution plan at DB2 II server and at remote sources



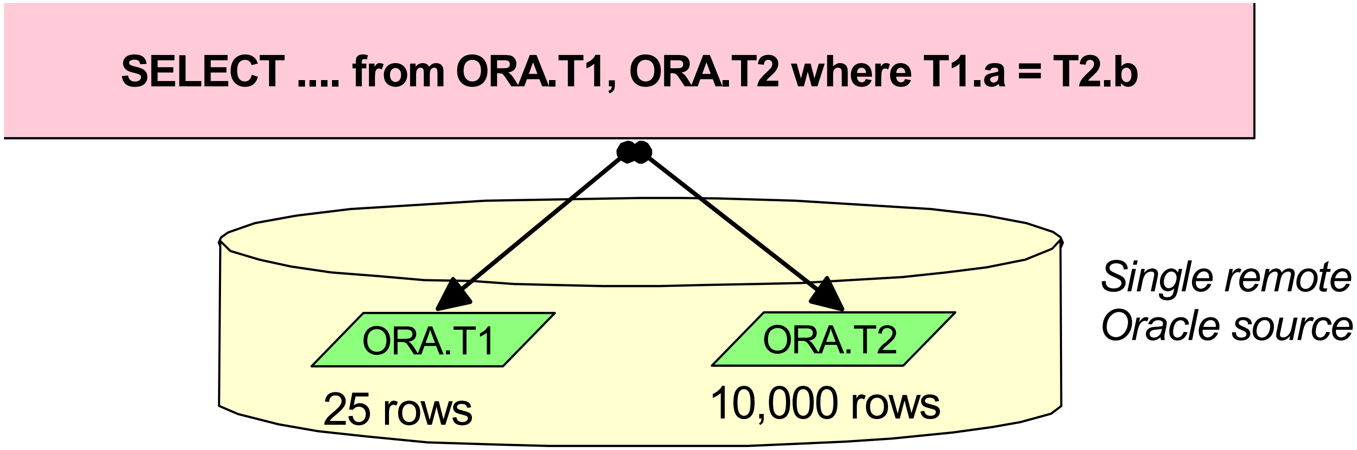
'Pushdown' of query operations

- DB2 II decides whether some or all parts of a query can be "pushed-down", i.e. processed at the remote data source(s). Pushdown-ability depends on
  - availability of needed functionality at remote source
  - server options (example: is collating sequence at Federated server and remote source the same?)
- Example: A remote source that can handle an equality predicate, but not count(*)....

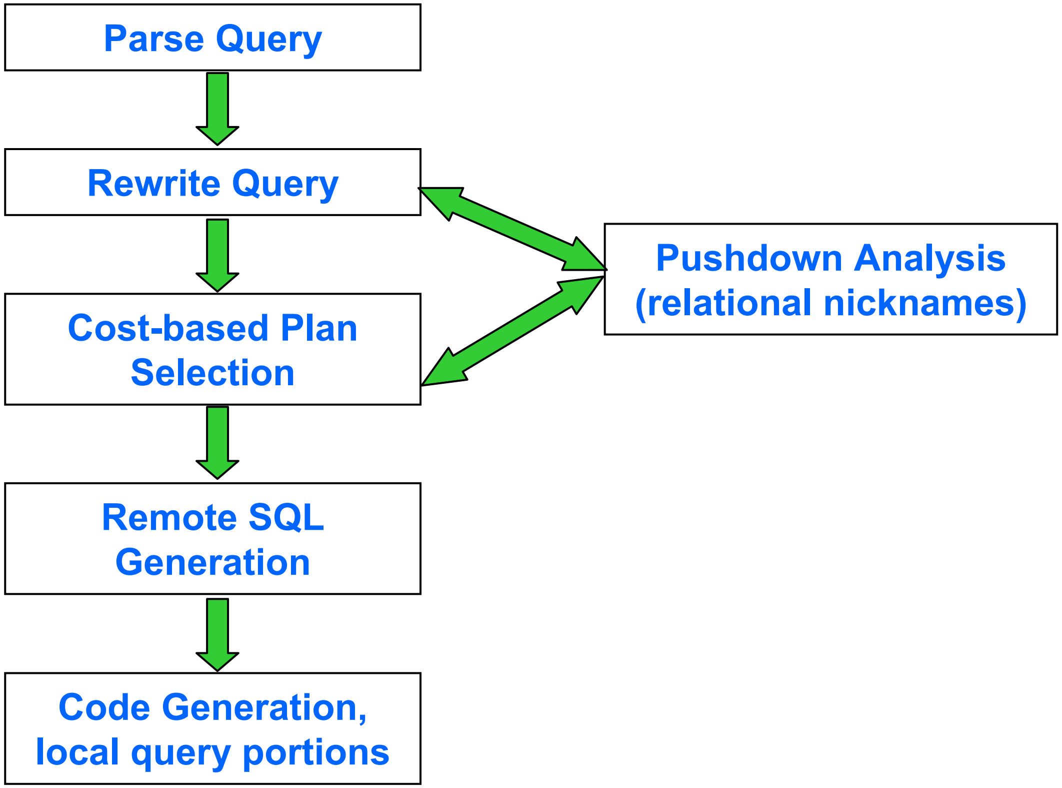


Actual pushdown is cost-based

- Just because processing can be pushed down doesn't mean it will be. Decision influenced by estimates of rows processed/returned.
- Consider a join of two nicknames ORA.T1 and ORA.T2 on a single remote source that is "nearly" a Cartesian product. May be better to do the join at the DB2 II server to avoid retrieval of many rows.
- Retrieving (10,000 + 25) rows to do a local join is probably faster than retrieving (10,000 * 25) = 250,000-row remote join result


SELECT .... from ORA.T1, ORA.T2 where T1.a = T2.b
ORA.T1
25 rows
ORA.T2
10,000 rows
Single remote Oracle source

DB2 II query optimizer flow for federated queries



Parse Query
Rewrite Query
Cost-based Plan Selection
Remote SQL Generation
Code Generation, local query portions
Pushdown Analysis (relational nicknames)

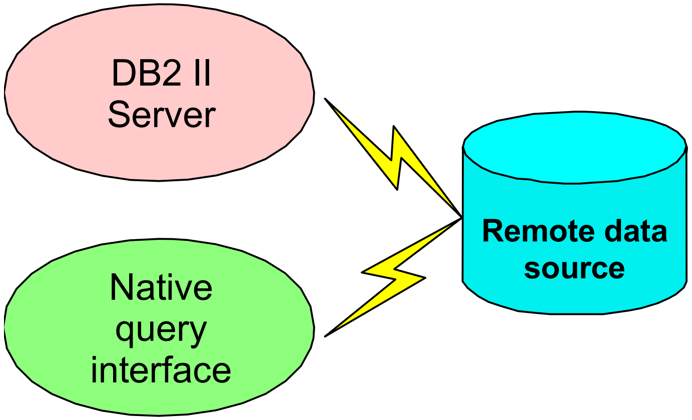
- Pushdown analysis is only part of the DB2 II optimizer's job! Must also:
  - Rewrite user queries (syntactic transformations)
  - Select best local execution plan
  - Generate correct SQL for remote query fragments (relational sources)

Agenda

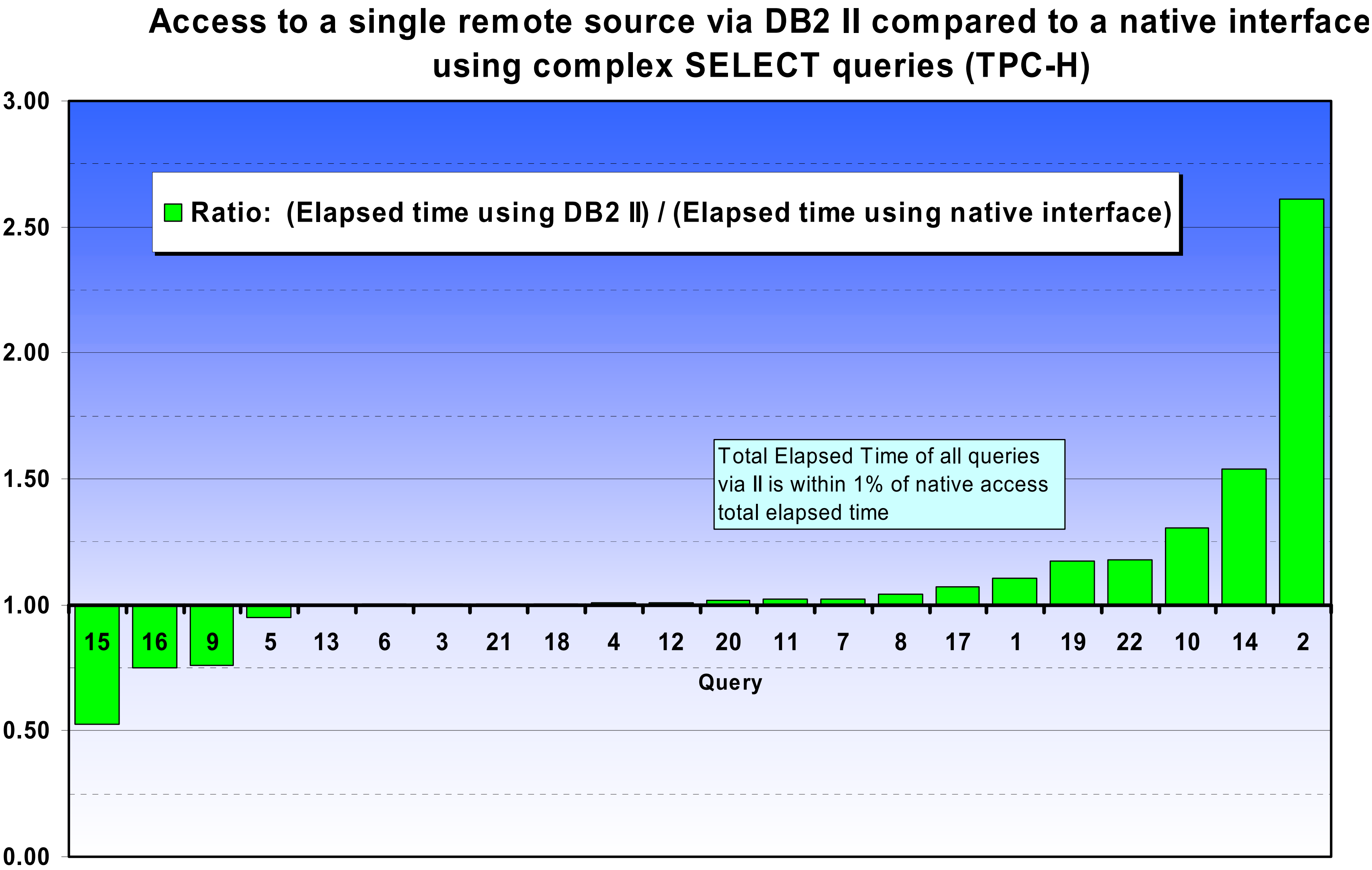
- DB2 Information Integrator basics
  - Basic components, how it works, pushdown, what impacts performance?
- Federated queries against a single remote source
  - When is performance "good?" Reading an EXPLAIN. Diagnosing performance problems.
- Federated queries involving multiple sources
  - What does good performance mean? Useful comparisons
- Configuring DB2 II for best performance
  - Server options, type and function mappings. Statistics. MQTs.
- New features in II V8.2 that can improve performance
  - Parallel execution in SMP and MPP environments
  - Fenced wrapper, informational constraints, better monitoring
- Using DB2 II as a part of a solution
  - "Appropriate deployment" – when should you use it?
  - Know your workload! Complement DB2 II with caching/replication

When is performance of a federated query “good?”

- Consider federated queries that involve
  - Only **one** remote source
  - No local data on the DB2 II instance
- Compare DB2 II performance with that of *native query interface* to the same remote source.
- Performance of a query is “good” if execution time of the following is about the same:
  - A DB2 II query against nicknames to remote objects
  - The same query using a native interface to the same remote objects
- Experiments using 22 TPC-H queries (complex decision support) with DB2 II



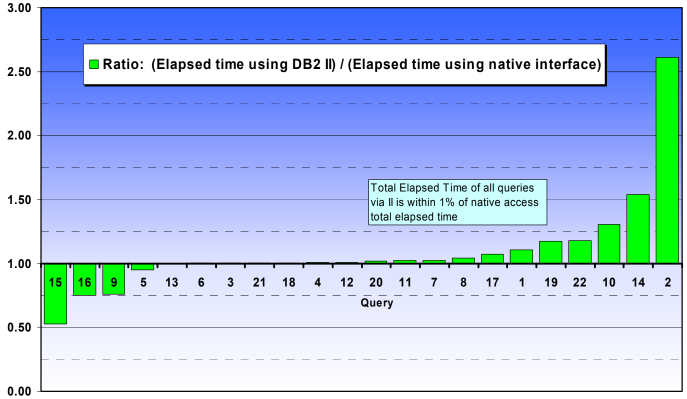
Experimental results



Total Elapsed Time of all queries via II is within 1% of native access total elapsed time

Analyzing individual queries (single remote source)

Access to a single remote source via DB2 II compared to a native interface using complex SELECT queries (TPC-H)



Total Elapsed Time of all queries via II is within 1% of native access total elapsed time

- Q4 took about the same time in both cases
- Q2 took more than twice as long using DB2 II as with the native interface. Q14 and Q10 took longer, too. Why?
- Queries 9, 16 and 15 ran faster with DB2 II. How can that happen?
- We can learn something from each query!

First step: Obtain DB2 II query execution plan

- Use Visual Explain or DB2EXFMT
- How to use DB2EXFMT:
  - Prerequisite: Create EXPLAIN tables by
    - Connecting to your database
    - db2 -tvf /sqllib/misc/EXPLAIN.DDL
- In DB2 CLP: "explain plan for <text_of_query>"
- This populates the EXPLAIN tables
- db2exfmt -d <dbname> -1 -o <output file>
- Reads the EXPLAIN tables and formats the query just explained
- EXPLAINs for federated queries are like other DB2 explains, but include "SHIP" operators designating interaction with remote sources.

How to read a DB2EXFMT explain plan

- Explain is a tree of **operators**, always in capital letters
- Each operator has estimates of rows output (above) and costs (below). The RETURN operator provides a legend
- Common operators:
  - NLJOIN, HSJOIN, MGJOIN
  - SCAN, IXSCAN, FETCH
  - SORT, GRPBY, DISTINCT, UNION
- The special SHIP operator moves processing to a different source.
- Everything above topmost SHIP is processed locally (SELECT queries)
- The leaves of the tree are tables, indexes, or nicknames ("NK")

```
          Rows
         RETURN
         (   1)
          Cost
           I/O
            |
         302400
         GRPBY
         (   2)
         753021
         208420
            |
         302400
         SORT
         (   4)
         748000
         202391
            |
         302400
         SHIP
         744245
         188236
         (   5)
     /------+-----\
 600000          2.39966e+07
NK: TPCD        NK: TPCD
CUSTOMER        LINEITEM
```

What part of my query is being pushed down?

For SELECT statements: everything above a SHIP operator is processed locally at the DB2 II server

```
               Rows
              RETURN
              (   1)
               Cost
                I/O
                |
                1
              SHIP
              (   2)
              208350
              33852.9
          /------+-----\
     600000             2.39966e+07
  NK:  TPCD              NK:  TPCD
CUSTOMER                 LINEITEM
```

This two-table join is completely pushed down to the remote source

```
               Rows
              RETURN
              (   1)
               Cost
                I/O
                |
              302400
              GRPBY
              (   2)
                |
              302400
              SORT
              (   4)
                |
              302400
              SHIP
              (   5)
          /------+-----\
     600000               2.39966e+07
  NK:  TPCD              NK:  TPCD
CUSTOMER               LINEITEM
```

This two-table join is pushed down, but the final sort/groupby is done at the DB2 II server

Reading the Explain plan for Q4

- Q4: join between nicknames to two tables (**Orders** and **Lineitem**) on the remote source
- Orders: 6M rows, Lineitem: ~24M rows
- No processing at the Federated server - topmost operator is SHIP. Query completely pushed down.
- What statement is SHIPped to the remote source? Look in details of the SHIP operator in EXPLAIN output
- 5 rows are estimated to be returned to the Federated server
- Complete pushdown + few rows returned ⇒ near-native performance

```
              Rows
              RETURN
             (   1)
               Cost
                I/O
                 |
                 5
               SHIP
              (   2)
           4.51969e+07
           1.80339e+06
          /------+-----\
   6e+06               2.39966e+07
NK: TPCD                 NK: TPCD
ORDERS                   LINEITEM
```

Q4: What is shipped to the remote source?

Original Query using nicknames:

```
SELECT O_ORDERPRIORITY, COUNT(*) AS
ORDER_COUNT FROM TPCD.ORDERS
WHERE
   O_ORDERDATE >= DATE('1993-07-01') AND
   O_ORDERDATE < DATE('1993-07-01')  + 3
   MONTHS
AND EXISTS
   (SELECT *  FROM TPCD.LINEITEM
   WHERE L_ORDERKEY = O_ORDERKEY
   AND L_COMMITDATE < L_RECEIPTDATE)
GROUP BY O_ORDERPRIORITY
ORDER BY O_ORDERPRIORITY;
```

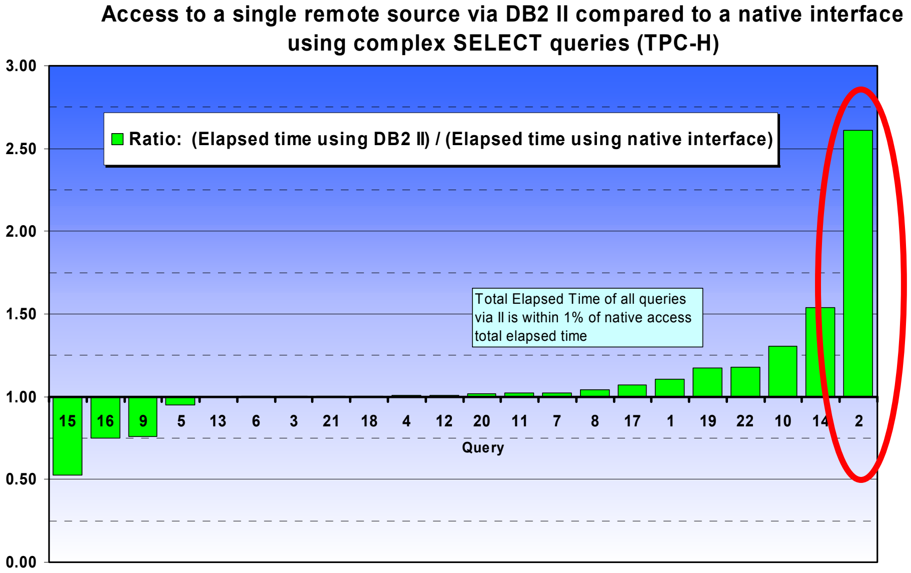
- Even for a completely pushed down query, the remote query text is not the same as the original due to
  - SQL differences on remote source
  - Federated optimization includes a rewrite phase!

Query shipped to the remote source

```
SELECT A0."O_ORDERPRIORITY",
COUNT(*) FROM "TPCH"."ORDERS" A0
WHERE (EXISTS
   (SELECT A1."L_RETURNFLAG" FROM
   "TPCH"."LINEITEM" A1 WHERE
    (A1."L_ORDERKEY" =
A0."O_ORDERKEY") AND
(A1."L_COMMITDATE" <
A1."L_RECEIPTDATE")))
AND
(TO_DATE('19930701
000000','YYYYMMDD HH24MISS') <=
A0."O_ORDERDATE")
AND (A0."O_ORDERDATE" < TO_DATE(
'19931001 000000','YYYYMMDD
HH24MISS'))
GROUP BY A0."O_ORDERPRIORITY"
ORDER BY 1 ASC;
```

Analyzing single-source queries, continued: Q2

```
SELECT S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS,
S_PHONE, S_COMMENT
FROM ORA1.PART, ORA1.SUPPLIER, ORA1.PARTSUPP, ORA1.NATION, ORA1.REGION
WHERE
P_PARTKEY = PS_PARTKEY AND
S_SUPPKEY = PS_SUPPKEY AND
P_SIZE = 15 AND
P_TYPE LIKE '%BRASS%' AND
S_NATIONKEY = N_NATIONKEY AND
N_REGIONKEY = R_REGIONKEY AND
R_NAME = 'EUROPE' AND
PS_SUPPLYCOST =
   (SELECT MIN(PS_SUPPLYCOST)
    FROM ORA1.PARTSUPP,
    ORA1.SUPPLIER, ORA1.NATION,
    ORA1.REGION
    WHERE P_PARTKEY = PS_PARTKEY
    AND S_SUPPKEY = PS_SUPPKEY AND
    S_NATIONKEY = N_NATIONKEY AND
    N_REGIONKEY = R_REGIONKEY AND R_NAME = 'EUROPE')
ORDER BY S_ACCTBAL DESC, N_NAME,S_NAME,P_PARTKEY
FETCH FIRST 100 ROWS ONLY
```


Access to a single remote source via DB2 II compared to a native interface using complex SELECT queries (TPC-H)

Total Elapsed Time of all queries via II is within 1% of native access total elapsed time

- 5-table join
- LIKE predicate

Explain for Q2 (simplified)

```
                                   Rows
                                  RETURN
                                  (   1)
                                    |
                                  TBSCAN
                                  (   2)
                                    |
                                  SORT
                                  (   3)
                                    |
                                  1943
      P_TYPE LIKE '%BRASS%' --->  FILTER
                                  (   4)
                                    |
                                  9043
                                  SHIP
                                  (   5)
                                    |
    +-----------------+-----------------+-----------------+---------------
--+
     5                25                40000             800000
3.2e+06
NICKNM: ORA1      NICKNM: ORA1      NICKNM: ORA1      NICKNM: ORA1
NICKNM: ORA1
   REGION            NATION           SUPPLIER             PART
PARTSUPP
```

Filtering locally means that potentially many more rows must be returned to DB2 II than if the predicate were applied at the remote source.

Local sorts are not necessarily bad. But if they are "reducing" it is usually better to do them remotely!

- The 5-way join is completely pushed down. But FILTER on **p_type** column and SORT operations are not
- LIKE predicates cannot be pushed down to Oracle – different semantics. Oracle assumes '%' at end of match string and DB2 does not
- Result: About 5 times more data moved to DB2 II than if LIKE/SORT pushed down.

Analyzing single-source queries, continued: Q14

SELECT 100.00 * SUM(
CASE WHEN P_TYPE LIKE 'PROMO%'
THEN L_EXTENDEDPRICE*(1-L_DISCOUNT)
ELSE 0 END) /
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT))
AS PROMO_REVENUE
FROM ORA1.LINEITEM, ORA1.PART
WHERE L_PARTKEY = P_PARTKEY AND
L_SHIPDATE >= DATE('1995-09-01') AND
L_SHIPDATE < DATE('1995-09-01') + 1 MONTH

```
                 Rows
                RETURN
                (   1)
                 Cost
                  I/O
                  |
                   1
                GRPBY
                (   2)
                93423.7
                16227.6
                  |
                254623
                SHIP
                (   3)
                93349.3
                16227.6
              /------+-----\
        800000              2.39966e+07
   NICKNM: ORA1          NICKNM: ORA1
        PART                 LINEITEM
```

Join is pushed down, but SUM/CASE is not. Why? LIKE predicate again!

What happens: Many rows must be shipped back to the DB2 II server

Analyzing single-source queries, continued: Q10

```
SELECT C_CUSTKEY, C_NAME, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
C_ACCTBAL, N_NAME, C_ADDRESS, C_PHONE,C_COMMENT
FROM ORA3.CUSTOMER, ORA3.ORDERS, ORA3.LINEITEM, ORA3.NATION
WHERE
C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY = O_ORDERKEY AND O_ORDERDATE >=
DATE('1993-10-01') AND O_ORDERDATE < DATE('1993-10-01') + 3 MONTHS
AND L_RETURNFLAG = 'R' AND C_NATIONKEY = N_NATIONKEY
GROUP BY C_CUSTKEY, C_NAME, C_ACCTBAL, C_PHONE, N_NAME, C_ADDRESS,
C_COMMENT ORDER BY REVENUE DESC FETCH FIRST 20 ROWS ONLY
```

```
                            Rows
                           RETURN
                           (   1)
                            Cost
                             I/O
                             |
                             20
                            SHIP
                           (   2)
                         3.95724e+06
                           642994
      +-------------------+-------------------+-------------------+
      25                600000               6e+06
  2.39966e+07
   NICKNM: ORA3       NICKNM: ORA3        NICKNM: ORA3        NICKNM: ORA3
       NATION           CUSTOMER               ORDERS             LINEITEM
```

Look at SHIPped query to see that FETCH FIRST n ROWS ONLY is not pushed down in this case!

Remote plan cannot take advantage of knowing that only a few rows are needed

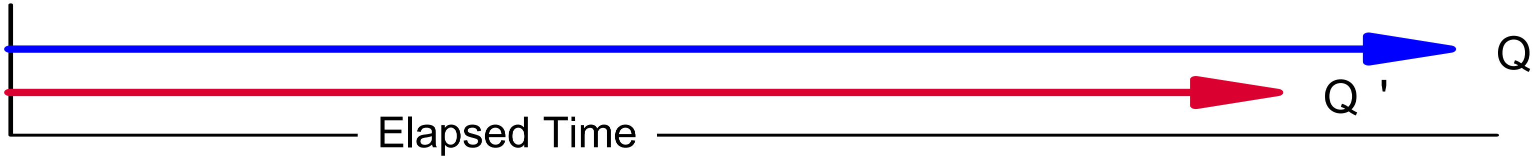
Analyzing single-source queries: Q15

- Complex 2 – table join with subquery
- Some predicates pushed down to each table, but some applied locally. Join done locally
- Even though the query is not pushed down, performance is better than via the native interface
- This is not common!
- May be due to tuning errors on the remote source

```
               RETURN
                 |
                1600
               TBSCAN
                 |
                1600
               SORT
                 |
                1600
               NLJOIN
          /--------+------\
        1                 1600
     GRPBY               FILTER
       |                   |
      38400               40000
     TBSCAN              HSJOIN
       |                /---+---\
      40000         40000        40000
     TEMP          SHIP         TBSCAN
       |             |            |
      40000         40000        40000
     SHIP     NICKNM: ORA1      TEMP
       |          SUPPLIER
   2.39966e+07
NICKNM: ORA1
    LINEITEM
```

Analyzing single-source queries: Summary

- Consider a federated query Q. Get DB2EXFMT output for Q.
- Watch for portions of the query that are not pushed down
- For a fully pushed-down query: Find the remote query Q' from the SHIP operator
- Try executing the remote query Q' directly at the remote source to isolate any performance issues (Easy: use DB2 II passthru feature). Try executing Q' with a SELECT COUNT(*) to verify the number of rows returned
- Compare execution time of Q at the Federated server and Q' at the remote source. If you know how, get an EXPLAIN of Q' at the remote source
- Useful tool: db2batch -d <dbname> -f <query file> -o p 2 executes query at Federated server and measures elapsed and CPU time
- In DB2 II V8.2: Can use enhanced snapshot monitoring to determine remote query time.



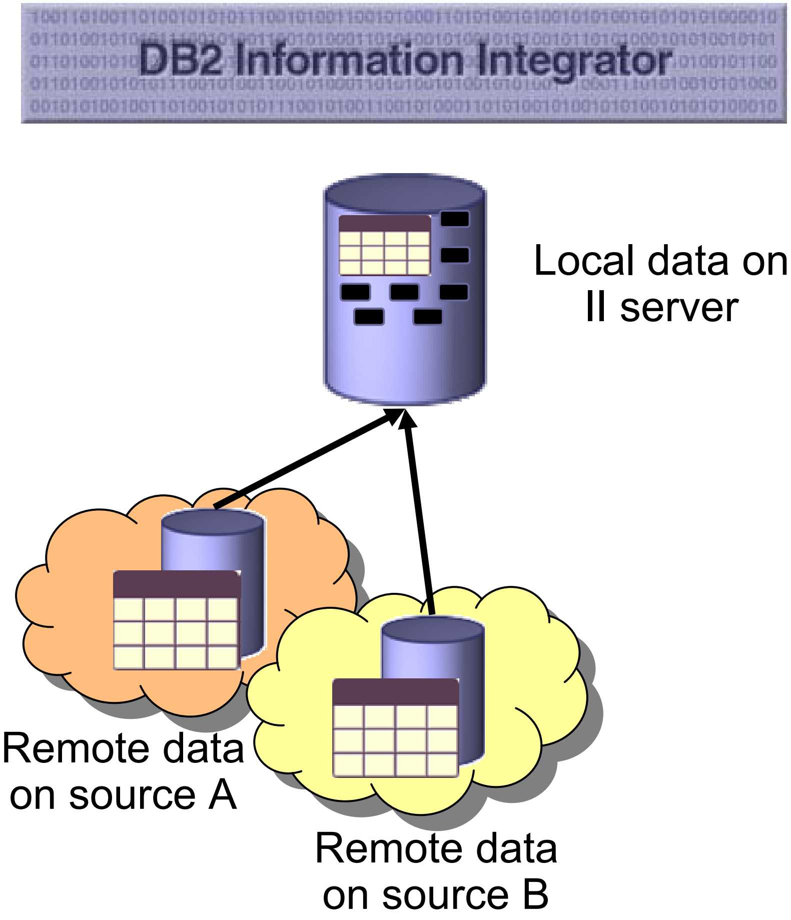
Performance of DB2 II against a single source

- DB2 II can be **comparable** with native access if
  - The query is completely pushed down,
  - Its [native-access] execution time is not trivial
  - The size of the result set returned is modest
- DB2 II will be **slower** than native access if
  - The query is not completely pushed down and large intermediate results must be moved to the DB2 II server
  - The query is very short (select… from <table> where <primary key> = ?)
  - The result set is large
- DB2 II can be **faster** than native access if
  - DB2 II rewrites a completely pushed-down query in an advantageous way or
  - The query is not completely pushed down and DB2 II chooses a good query execution plan for work done locally

Agenda

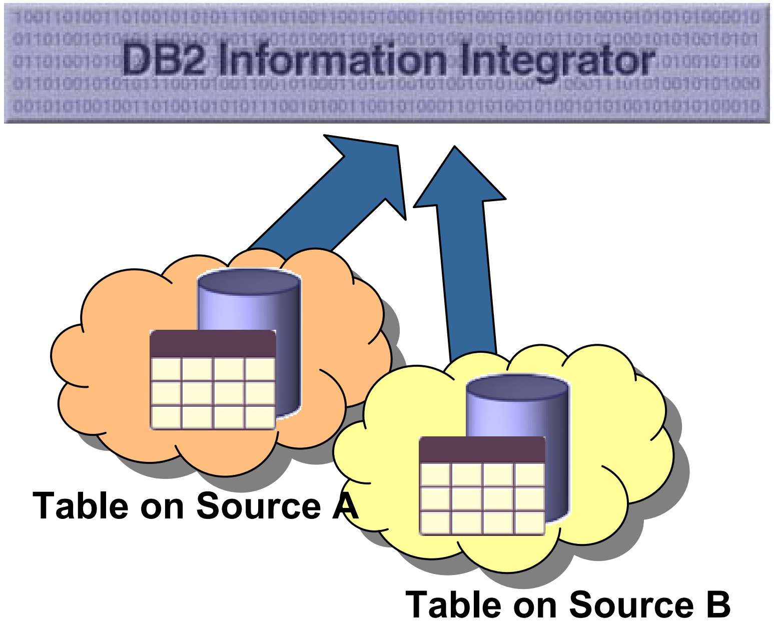
- DB2 Information Integrator basics
  - Basic components, how it works, pushdown, what impacts performance?
- Federated queries against a single remote source
  - When is performance "good?" Reading an EXPLAIN. Diagnosing performance problems.
- Federated queries involving multiple sources
  - What does good performance mean? Useful comparisons
- Configuring DB2 II for best performance
  - Server options, type and function mappings. Statistics. MQTs.
- New features in II V8.2 that can improve performance
  - Parallel execution in SMP and MPP environments
  - Fenced wrapper, informational constraints, better monitoring
- Using DB2 II as a part of a solution
  - "Appropriate deployment" – when should you use it?
  - Know your workload! Complement DB2 II with caching/replication

Performance of distributed (multi-source) queries



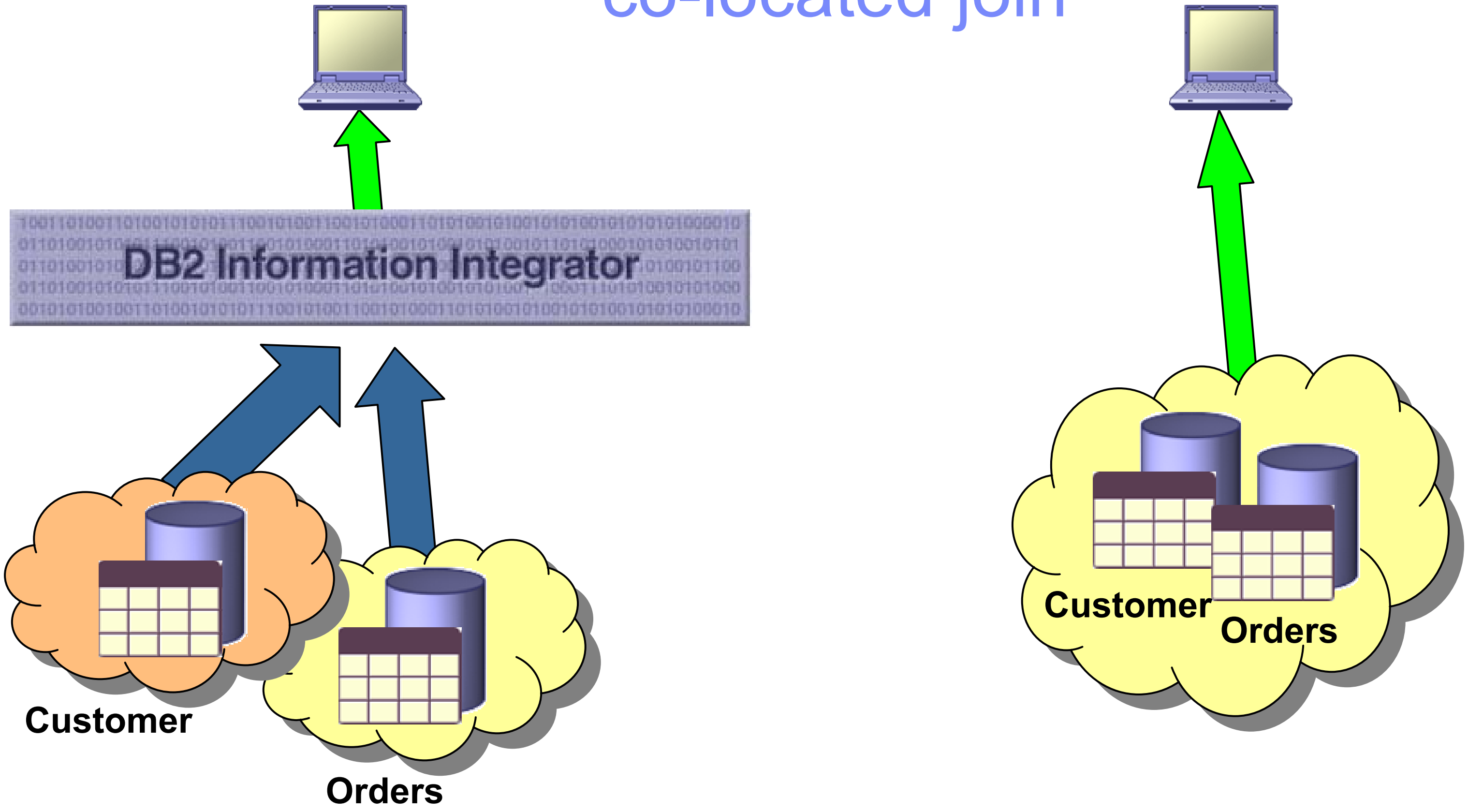
- Consider queries involving multiple sources (**distributed** queries)
  - Nicknames on more than one remote source –or-
  - Local DB2 II data and nicknames on one or more remote sources
- No such thing as “full pushdown” for distributed queries
- Some processing of the query must always be done by DB2 II.
- Key performance considerations:
  - How much data must be moved from the data sources to the DB2 II server?
  - How many interactions are there between DB2 II and each source?

Performance of a distributed join



- To join two remote tables on different sources, all rows *involved in the join* from both must be shipped to the DB2 II server
- When does such a join perform “well?”
- One possibility Compare distributed join with the same join using co-located data on a single remote database How much slower is the distributed join?
- Depends on patterns of data access
  - Favorable case: Even if both tables are huge, a small number of interesting rows can be filtered out at the remote source before joining.
  - Less favorable case: Must retrieve many rows from one or both tables to do the join

Performance of a distributed join: Comparison with co-located join



- Each Order contains a value and a reference to the Customer key that placed the order. Both tables have index on customer key.
- Consider possible joins between Customer and Orders.

Plans for two possible distributed joins between Customer and Orders

#1: Count orders for 14 specific customers identified by key. List customer and order count

```
            Rows
            RETURN
            (   1)
             Cost
              I/O
              |
              14
            GRPBY
            (   2)
              |
              140
            NLJOIN
            (   3)
          /----+---\
      14             10
     TBSCAN        SHIP
     (   4)        (   9)
       |              |
      14            6e+06
     SORT      NICKNM:  ORA1
     (   5)    ORDERS
       |
      14
     SHIP
     (   6)
       |
     600000
NICKNM:  ORA2
CUSTOMER
```

#2: Find count of customers for whom the total value of all orders placed is at least $4.5M.

```
             Rows
            RETURN
            (   1)
             Cost
              I/O
              |
           0.333333
            FILTER
            (   2)
              |
               1
            GRPBY
            (   3)
              |
          4.79999e+06
            HSJOIN
            (   4)
        /-------+-----\
 4.99999e+06            479999
   SHIP                 SHIP
   (   5)               (   7)
     |                     |
    6e+06                600000
NICKNM:  ORA1     NICKNM:  ORA2
ORDERS            CUSTOMER
```

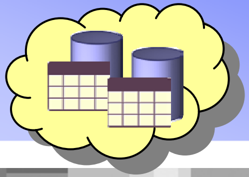
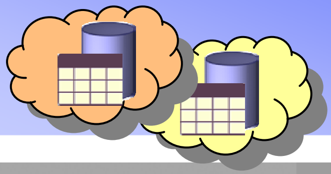
Reference: The actual queries

```
-- Query #1: Count orders for a few customers in a particular nation
-- Join links customer key with orders for that customer.
-- Favorable.  Indexed access to customer key on both sides

select c_name, c_phone, count(*) from tpcd1.orders, tpcd2.customer
where o_custkey = c_custkey
and c_nationkey = 4 and c_custkey between 1 and 500
group by c_name, c_phone order by c_name;
```

```
-- Query #2: Look up total value of orders for a large set of customers
-- Find and count customers that ordered more than $4.5M
-- Less favorable, many thousands of customers/orders must be retrieved

select count(*) from tpcd1.orders, tpcd2.customer
where o_custkey = c_custkey
and c_custkey < 500000 and c_nationkey <> 5 having sum(o_totalprice) > 4.5e+06;
```

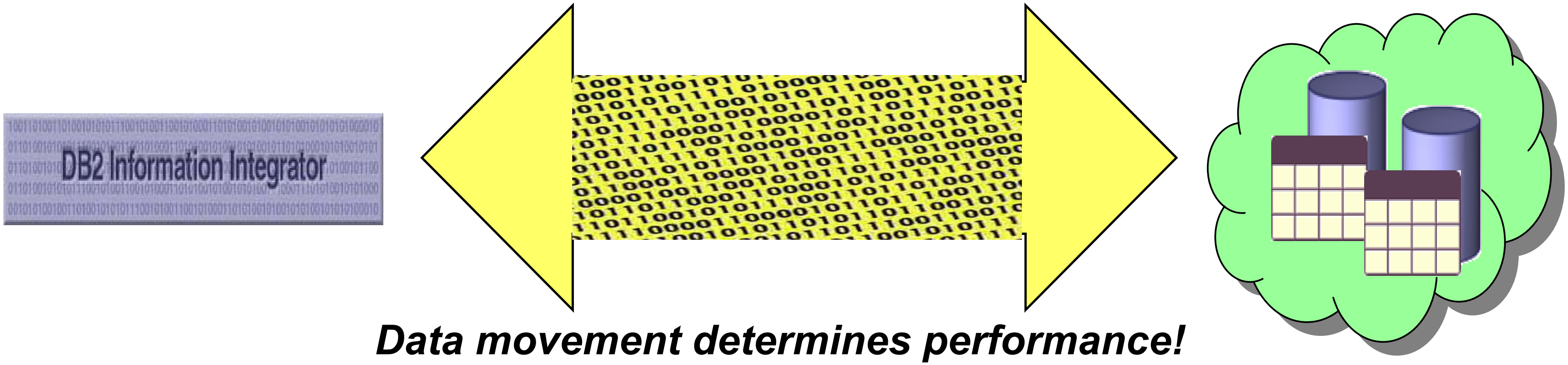
Performance of distributed queries compared with equivalent queries against consolidated data

- Run queries #1 and #2
  - With both tables on a single Oracle instance
  - As distributed queries over two Oracle instances, one table on each
- Compare! Numbers are close for #1, far apart for #2. Why?
  - #2 requires substantial data movement back to DB2 II server

Query	Both tables consolidated on one Oracle instance	Tables in separate Oracle instances, federated query using DB2 II
#1: Find count of matching Orders for 14 selected Customer keys	18 ms.	22 ms.
#2: From all 500K customers, count those having a total order value of > $4.5M this year.	11 sec 	106 sec 

Distributed join performance:

- Performance relative to an equivalent join against consolidated data depends on
  - (mostly) the **amount of data** that must be moved to DB2 II server!
  - (to a lesser extent) the **number of interactions** with a remote source required by the execution plan
- Will distributed joins perform well for you?
  - Depends on patterns of data access
  - Know your queries!

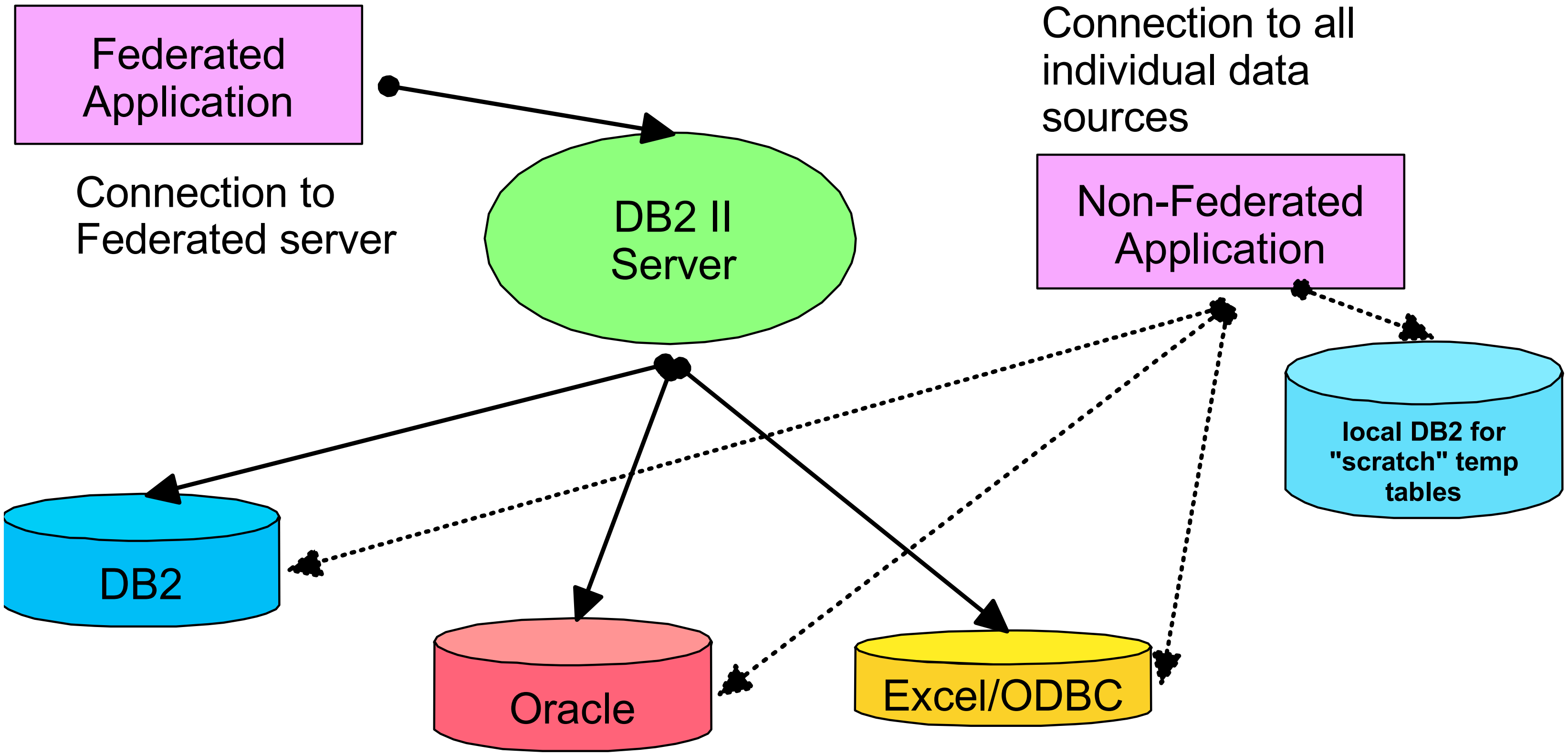


Data movement determines performance!

Another perspective: Application federation

- Comparison with consolidation is a bit “academic”. Consolidation may not be an option.
- How else could we judge distributed query performance?
- Compare with application federation
  - Implement distributed queries in an application that explicitly connects to multiple remote data sources
  - Have application connect to DB2 II and let it handle distributed queries
- Experimental results…

Comparing performance of distributed queries in a J2EE application with and without DB2 II


Federated Application
Connection to Federated server
DB2 II Server
Connection to all individual data sources
Non-Federated Application
local DB2 for "scratch" temp tables
DB2
Oracle
Excel/ODBC

Comparing performance of distributed queries in a J2EE application with and without DB2 II

- **Without DB2 II**: Application connects to each source, issues SQL in its dialect, retrieves appropriate data from each, inserts into local temporary tables, and processes query locally
- **With DB2 II**: Application connects only to II server and submits queries against nicknames to several sources (~40% less code is typical)
  - DB2 II manages the decomposition and processing of the query
  - Can also create join and union views over nicknames to make multiple remote tables appear as one to the application
- **Results** of experiments with J2EE servlets issuing queries involving three remote data sources

Query	Time using DB2 II	Time without DB2 II
1	3.5 sec	3.4 sec
2	0.24 sec	0.16 sec
3	54.2 sec	170.1 sec
4	6.5 sec	81.2 sec
5	15.1 sec	9.9 sec

Application federation conclusions:

- A hand-coded application that connects to all the sources may be able to outperform one that uses DB2 II but...
  - It's not easy to correctly implement the distributed query by decomposing it and merging results from each source locally
  - Managing multiple connections and SQL dialects is hard work
- DB2 II can outperform or at least be competitive with hand-coded “application federation”
  - DB2 II optimizer can make good choices for query plans if nickname statistics are kept up to date
  - DB2 II works with data in memory instead of temporary tables when possible

Agenda

- DB2 Information Integrator basics
  - Basic components, how it works, pushdown, what impacts performance?
- Federated queries against a single remote source
  - When is performance “good?” Reading an EXPLAIN. Diagnosing performance problems.
- Federated queries involving multiple sources
  - What does good performance mean? Useful comparisons
- Configuring DB2 II for best performance
  - Server options, type and function mappings. Statistics. MQTs.
- New features in II V8.2 that can improve performance
  - Parallel execution in SMP and MPP environments
  - Fenced wrapper, informational constraints, better monitoring
- Using DB2 II as a part of a solution
  - “Appropriate deployment” – when should you use it?
  - Know your workload! Complement DB2 II with caching/replication

DB2 Information Integrator configuration

- Good configuration, accurate knowledge of remote sources and objects is vital to best-performing execution plans!
- Server and column options
  - DB2 II needs to know what remote source can and can't do; must set **server options** correctly.
- Data type mappings
  - Be aware of mappings between remote and local data types
- Nickname statistics
- Index specifications
- Materialized Query Tables over nicknames
- "Normal" DB2 tuning (SORTHEAP, Bufferpools, etc.)

Server and column options affect performance!

- Set these on a per-server basis
  - COLLATING_SEQUENCE: Set to ‘Y’ if remote and local strings sort the same way.
  - DB2_MAXIMAL_PUSHDOWN: enforce cost-blind pushdown of query processing (safer: set per session!)
  - COMM_RATE, CPU_RATIO, IO_RATIO: Information about link speed, whether remote CPU/IO is slower or faster than at the DB2 II server
  - PLAN_HINTS: Have DB2 II generate plan hints for Oracle sources (not recommended)
- Set these on a per-column basis
  - VARCHAR_NO_TRAILING BLANKS: Does your Oracle VARCHAR data have trailing blanks? Set to ‘Y’ if you’re *sure* it doesn’t!
  - NUMERIC_STRING: Set to ‘Y’ if a character string contains only digits; then COLLATING_SEQUENCE is not an issue for this column

Effect of COLLATING_SEQUENCE server option

- Tells DB2 II whether strings sort the same way remotely or not
- If yes, it's safe to push down SORTs, character inequality predicates. Example: **`select count(*)… where p_name > 'F';`**

```
     RETURN
       |
        1
     GRPBY
       |
     593357
     FILTER
       |
     800000
     SHIP
       |
     800000
NICKNM:  ORA1
       PART
```

COLLATING_SEQUENCE = 'N'

'Y' setting enables pushdown of predicate evaluation and aggregation!

```
    RETURN
    (    1)
      |
      1
    SHIP
    (    2)
      |
    800000
NICKNM:  ORA1
      PART
```

COLLATING_SEQUENCE = 'Y'

Effect of DB2_MAXIMAL_PUSHDOWN setting

- Can be set per server
  - As a server option (affects ALL queries to that server!) or…
  - For the duration of a session (better!)
- All processing that can be pushed down safely will be
  - Even if the optimizer disagrees
- Use only when you must to obtain desired pushdown
- Careful, can backfire and cause worse performance!
  - Example: Join A1, A2 from source A with B1 from source B
  - Suppose: (A1 join B1) is reducing, (A1 join A2) is “exploding”
  - Best join order is: (A1 join B1) join A2
  - DB2_MAXIMAL_PUSHDOWN will force (A1 join A2) join B2

Effect of VARCHAR_NO_TRAILING BLANKS

- Oracle considers trailing blanks in character comparisons, DB2 doesn’t. DB2 II compensates for this difference.
- Unless… you tell it that a column has no trailing blanks!
- ‘Yes’ means ‘No’, and ‘No’ means ‘Yes!’
  - ‘Y’ means there are NO trailing blanks in this column’s data
  - ‘N’ means there could be trailing blank in this column’s data (default)
- We trust you…
- Setting to ‘N’ is always safe, but may not perform as well
- Setting to ‘Y’ may
  - Yield much better performance
  - Get the wrong answer!!! (If the data really *does* have trailing blanks)

Effect of VARCHAR_NO_TRAILING_BLANKS

select count(*) from ora1.part where p_name > 'cornflower';

```
         Rows
        RETURN
       (   1)
         Cost
          I/O
          |
           1
        SHIP
       (   2)
           |
        800000
   NICKNM: ORA1
         PART
```

- DB2 II gets plan on left regardless of setting
- If VARCHAR_NO_TRAILING_BLANKS is ‘Y’, following statement is pushed down:
  - SELECT COUNT(*) FROM "TPCH"."PART" A0 WHERE ('cornflower' < A0."P_NAME")
  - If trailing blanks do exist.. No matches.. Wrong answer!
- If VARCHAR_NO_TRAILING_BLANKS is ‘N’, DB2 II pushes down:
  - SELECT COUNT(*) FROM "TPCH"."PART" A0 WHERE (RPAD('cornflower',55,' ') < RPAD(A0."P_NAME",55,' '))
  - Blank padding assures correct results. But….
  - Remote index on P_NAME rendered useless! Slow.
- Use with care!

Column type mappings between remote and local columns may have performance impact

- Example: Oracle DATE type is mapped to DB2 TIMESTAMP type by default;
  - … WHERE <nickname col of type TIMESTAMP> = DATE(‘1996-04-01’) won’t work!
  - WHERE CAST(<nickname col> as DATE) = DATE(‘1996-04-01’) works, but may push down CAST and result in poor performance
- Can override defaults by altering local nickname column types if appropriate. May help performance
  - ALTER NICKNAME <nn> ALTER COLUMN <colname> LOCAL TYPE DATE;
  - You are telling DB2 that the remote Oracle DATE column should be mapped to a local DATE column
  - Implicitly, you’re telling us that the “timestamp” portion of the Oracle “DATE” is zero – and we trust you.
  - Comparisons of that nickname column with DATE literals work –no CAST!

Nickname statistics – where do they come from?

- DB2 II retrieves statistics from remote-source catalog to populate DB2 catalog at CREATE NICKNAME time
  - Nickname statistics are only as good as what's stored on remote!
  - Statistics changes are *not* automatically propagated to the DB2 II server
  - Do Runstats equivalent on remote system *before* creating nicknames
- Nicknames over remote views, non-relational objects, and aliases/synonyms have *no* statistics by default
- If your EXPLAIN looks like this, statistics for your nicknames are probably empty (1000 rows is default rowcount!)

```
                                 Rows
                                RETURN
                                (   1)
                                   |
                                   20
                                 SHIP
                                (   2)
                                   |
     +-----------------+---------+--------+-----------------+
    1000              1000              1000              1000
NICKNM: ORA3      NICKNM: ORA3      NICKNM: ORA3      NICKNM: ORA3
     NATION           CUSTOMER           ORDERS          LINEITEM
```

Nickname statistics: What does DB2 II collect?

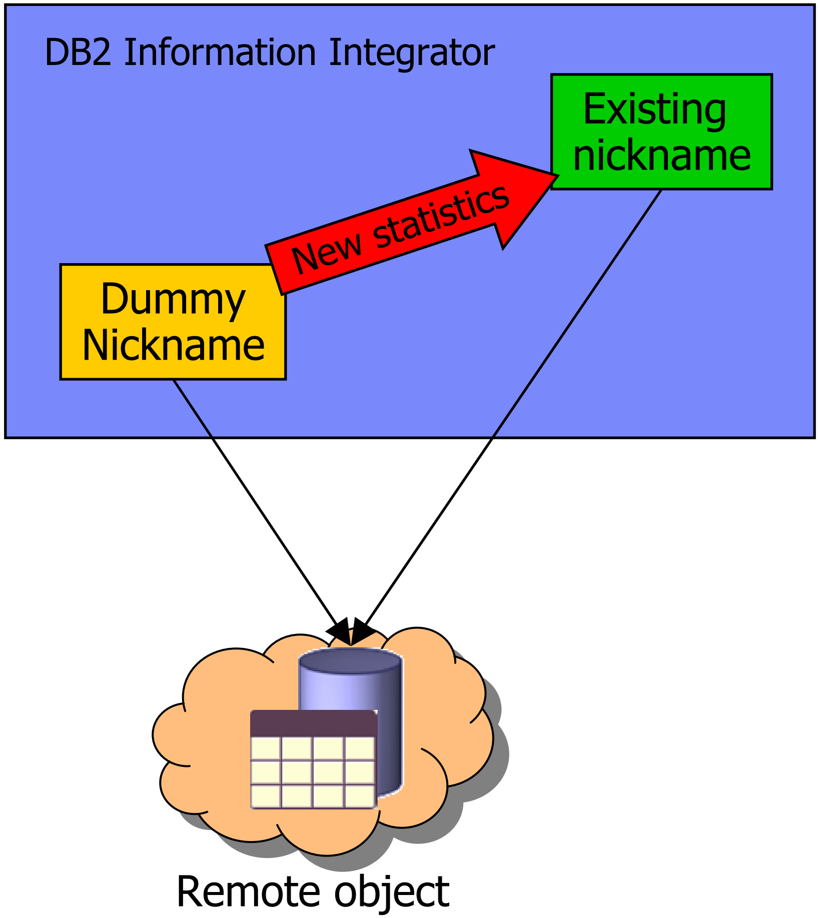
Statistic:	MSSQL	Informix (IDS)	Oracle (Net8)	Sybase CTLIB	DRDA: UDB	Teradata	DRDA: z/OS	DRDA: AS400
card		X	X	X	X	X	X	
npages		X	X	X	X	X	X	
fpages		X	X		X	X	X	
overflow			X		X			
colcard		X	X		X		X	
high2key		X			X			
low2key		X			X			
firstkeycard		X	X		X		X	
fullkeycard	X		X		X		X	
nlevels		X	X		X		X	
nleaf	X	X	X		X		X	
clusterratio		X	X		X		X	

- Different levels of statistics are retrieved for different sources
- Some of these are in the process of being improved
- Some of them are just plain hard to find!
- No stats for remote views, aliases, nonrelational objects

Nickname statistics: Keeping them up-to-date

- Keep nickname statistics up-to-date after CREATE NICKNAME by doing one of:
  - Drop and recreate nickname initiates stats retrieval (not always practical)
  - Use new NNSTAT() stored procedure (DB2 II V8.2)
  - Control Center "Update Statistics" facility (calls NNSTAT()) (DB2 II V8.2)
- All of the above merely **re-retrieve remote source statistics**. DB2 II does not do its own statistics gathering for nicknames.
- RUNSTATS doesn't work for nicknames (yet).
- What about statistics for nicknames over remote views, aliases, nonrelational objects? Can use get_stats tool, downloadable at
  - http://www-106.ibm.com/developerworks/db2/downloads/getstats/
  - Issues queries directly against nicknames to populate statistics
  - Can be very resource-intensive. Use with care.

The NNSTAT() stored procedure / "Update Statistics" for nicknames in the Control Center



- Calling NNSTAT() for an existing nickname
- Creates a temporary dummy nickname to the same remote object
- CREATE NICKNAME retrieves statistics from remote source
- "Fresh" statistics from dummy transferred to existing nickname, dummy deleted
- If *no* statistics retrieved then NNSTAT() runs get_stats under-the-covers! (Query-based statistics collection)

Nickname statistics: How do I know what I’ve got?

- Check nickname statistics in
  - SYSSTAT.COLUMNS: "colcard", "high2key", and "low2key"
  - SYSSTAT.TABLES: "card"
  - SYSSTAT.INDEXES: "firstkeycard", "fullkeycard"
- If statistics are missing for a nickname, and are not provided by NNSTAT(), you can supply values based on knowledge of remote source statistics
- Improved statistics can enable better pushdown and plan decisions

Check column statistics for nickname:

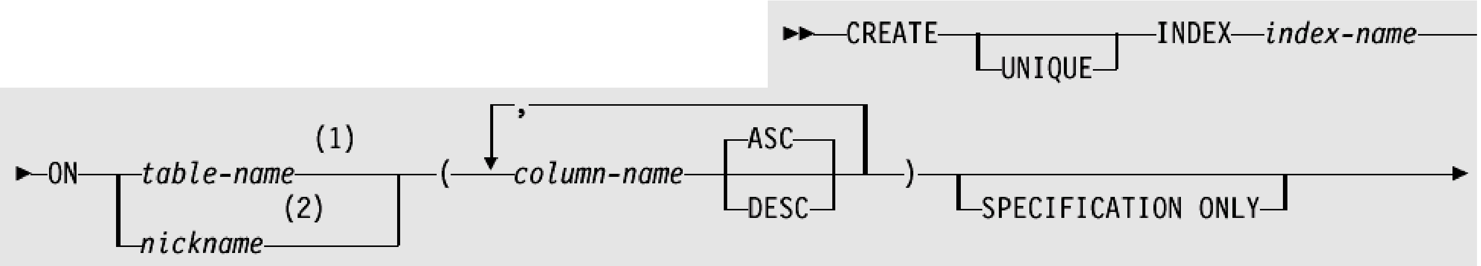
```
select char(colname,20) as colname,
colcard, char(high2key, 33) as high,
char(low2key, 33) as low
from sysstat.columns
where tabschema = ‘<schema>' and
tabname = ‘<table_name>';
```

Fix column statistics for a nickname column:

```
update sysstat.columns
set colcard=‘2526’,
    high2key = '1998-11-30',
    low2key   = '1992-01-03'
where colname = 'L_SHIPDATE' and
tabname = 'LINEITEM' and
tabschema in ('TPCD');
```

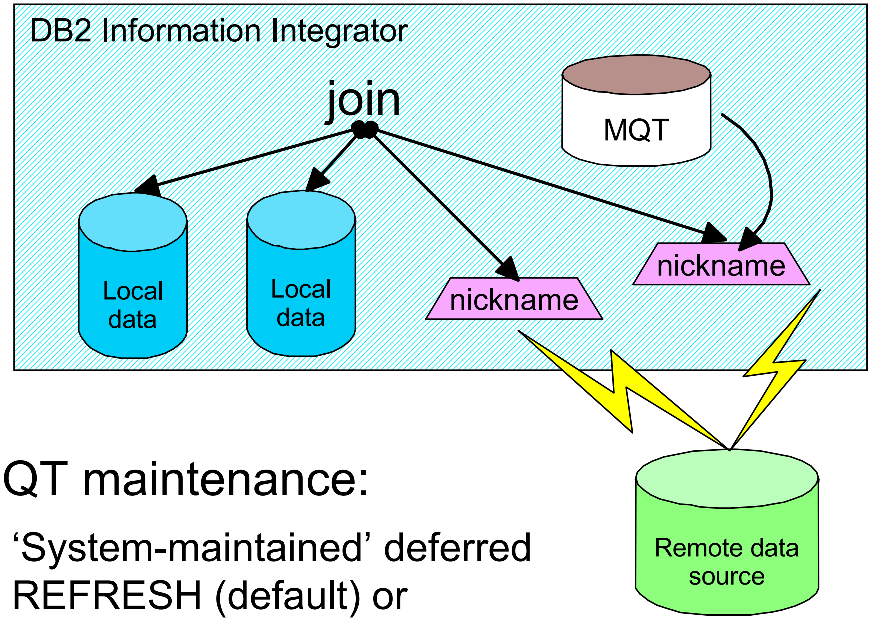
Indexes on nicknames and index specifications

- There are no actual "local indexes" on nicknames. Information on remote indexes is kept in the DB2 II catalog
- Normally: Information about remote indexes is picked up during nickname creation. DB2 II doesn't know about
  - indexes added on the remote source after nickname creation
  - Indexes "underneath" remote nicknamed objects such as:
    - views
    - synonyms (i.e. in Informix)
    - nonrelational objects
- An <u>index specification</u> lets you tell (or lie to) DB2 II about an index or access path on a nicknamed remote object



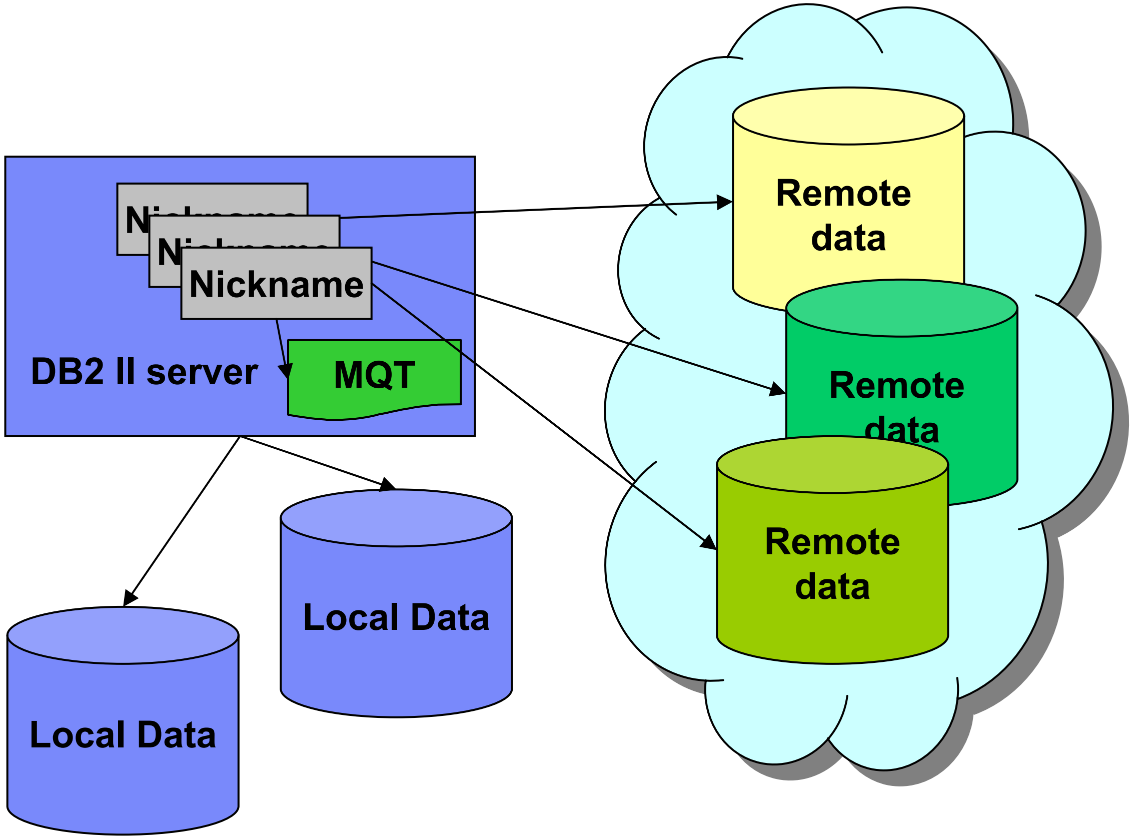
Local caching: Materialized query tables (MQTs) over nicknames

- MQT: local table defined by the result of a query
  - Can include joins, aggregations over multiple nicknames
  - Can be indexed, replicated in partitioned environment
  - Optimizer "routes to" them transparently as appropriate
  - DB2 II V8.1: can include both local DB2 tables and nicknames
  - DB2 II V8.2: can include nicknames to nonrelational objects
- Use to replace remote access with local access



- MQT maintenance:
  - 'System-maintained' deferred REFRESH (default) or
  - 'User-maintained', keep in sync via replication
  - Simplified setup for nickname caches with replication in DB2 II V8.2

Scenarios that benefit from MQTs over remote data



- Local fact table, remote dimension tables: Cache a local copy of a small-to-medium dimension table to save on remote access
- Cache a prejoin of a remote fact table with one or more remote dimensions to have a local copy of only the interesting part of a fact table
- Cache an interesting aggregate of a remote fact table with one or more dimensions and satisfy many queries without remote access

DB2 II database and instance configuration

- Tune “as usual” based on local data at DB2 II instance first
- Primary extra resources needed by federated queries are CPU and memory!
- Nicknames are not associated with any tablespace so have no bufferpool requirements of their own
- Federated queries that are not completely pushed down may need
  - SORTHEAP (adjust SHEAPTHRES based on concurrency)
  - Temporary table space (with associated bufferpool)
- Help with configuration:
  - “**Data Federation with IBM DB2 Information Integrator V8.1**” Redbook, see www.redbooks.ibm.com, SG240752
  - **Federated Systems Guide** (DB2 II V8.2 product documentation)

Agenda

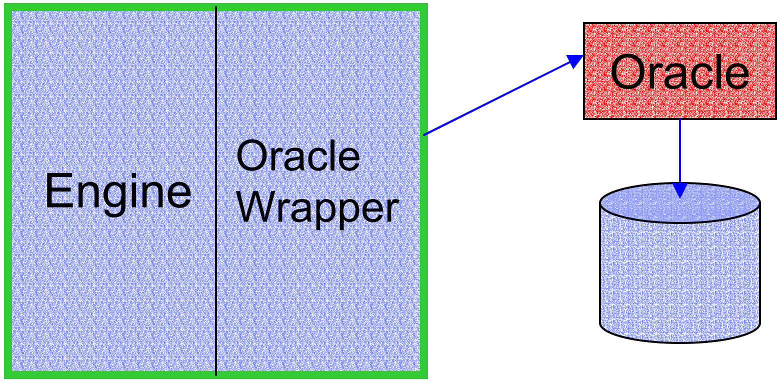
- DB2 Information Integrator basics
  - Basic components, how it works, pushdown, what impacts performance?
- Federated queries against a single remote source
  - When is performance “good?” Reading an EXPLAIN. Diagnosing performance problems.
- Federated queries involving multiple sources
  - What does good performance mean? Useful comparisons
- Configuring DB2 II for best performance
  - Server options, type and function mappings. Statistics. MQTs.
- New features in II V8.2 that can improve performance
  - Parallel execution in SMP and MPP environments
  - Fenced wrapper, informational constraints, better monitoring
- Using DB2 II as a part of a solution
  - “Appropriate deployment” – when should you use it?
  - Know your workload! Complement DB2 II with caching/replication

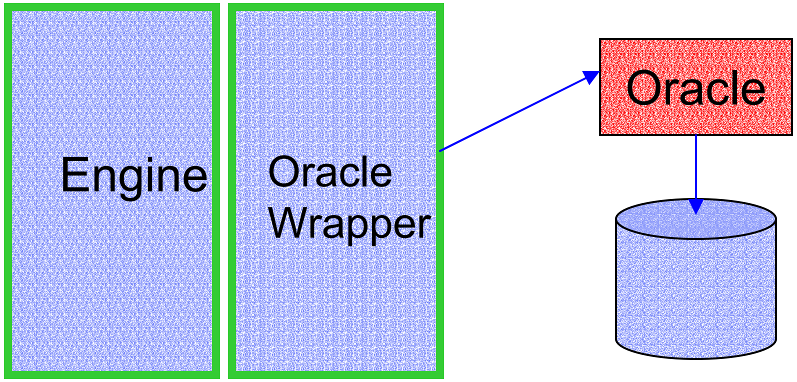
What’s new for performance in DB2 II V8.2

- Fenced wrapper. Benefits, Impact
- Better integration of nicknames into parallel query plans
  - In partitioned databases (MPP). When are the new plans beneficial?
  - In non-partitioned databases (SMP) with INTRA-PARALLEL enabled
  - Measurement results for SMP and MPP
- UNION ALL processing enhancements
- Informational constraints over nicknames
- Enhanced snapshot monitoring capabilities for federated queries

Fenced wrappers

- Today, all wrappers are “trusted”
  - Wrappers run in same process as DB2 engine
  - Maximal efficiency, maximal danger
- In DB2 II V8.2, wrappers may run “fenced”
  - Explicit choice when (any) wrapper is created or via “ALTER WRAPPER”
  - Allows isolation of wrappers
    - Protects engine, local data
    - Eases problem determination
    - Good for 3rd party wrappers, wrapper development
  - Allows resource sharing across apps for scalability via threading
  - Some cost in performance
  - Potential exploitation of MPP parallelism in partitioned systems

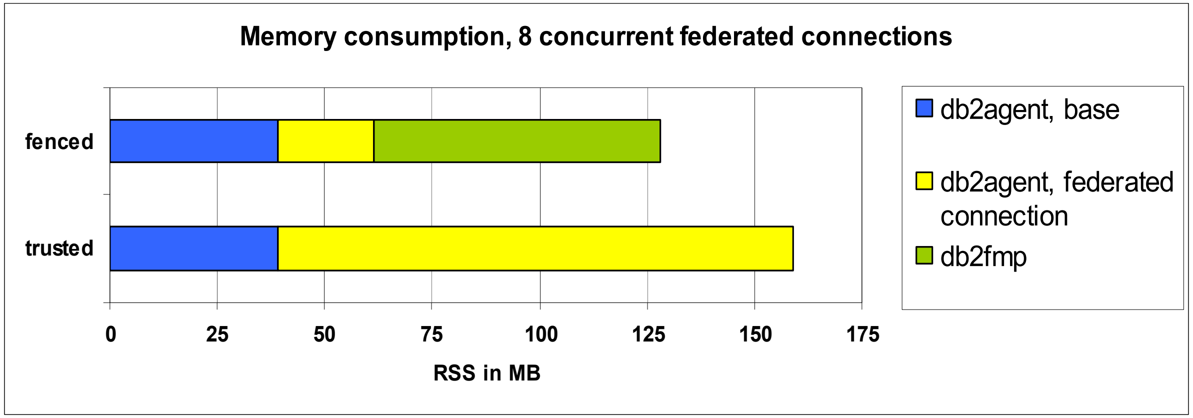




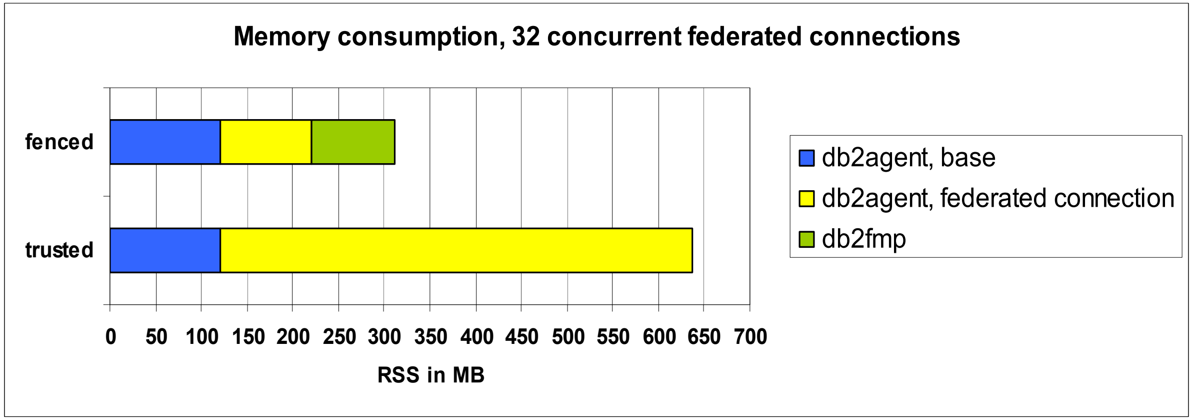
Benefits and costs of the fenced wrapper

- Memory savings via shared use of wrapper code
- Fenced wrapper enables parallel plans in MPP environment
- Increased cost (CPU) due to extra process hop!
- Elapsed time impact in queries that don't benefit from parallel execution
  - Not so much for bulk fetches (< 5-10%)
  - Very noticeable for nested join with nickname inners (Up to 50%!)
- When to use the fenced wrapper
  - To enable parallelism in MPP environments
  - To save memory in a high-concurrency environment
  - Safety and fault isolation

Memory savings due to use of fenced wrapper: Comparing trusted and fenced for 8 and 32 connections sharing a wrapper

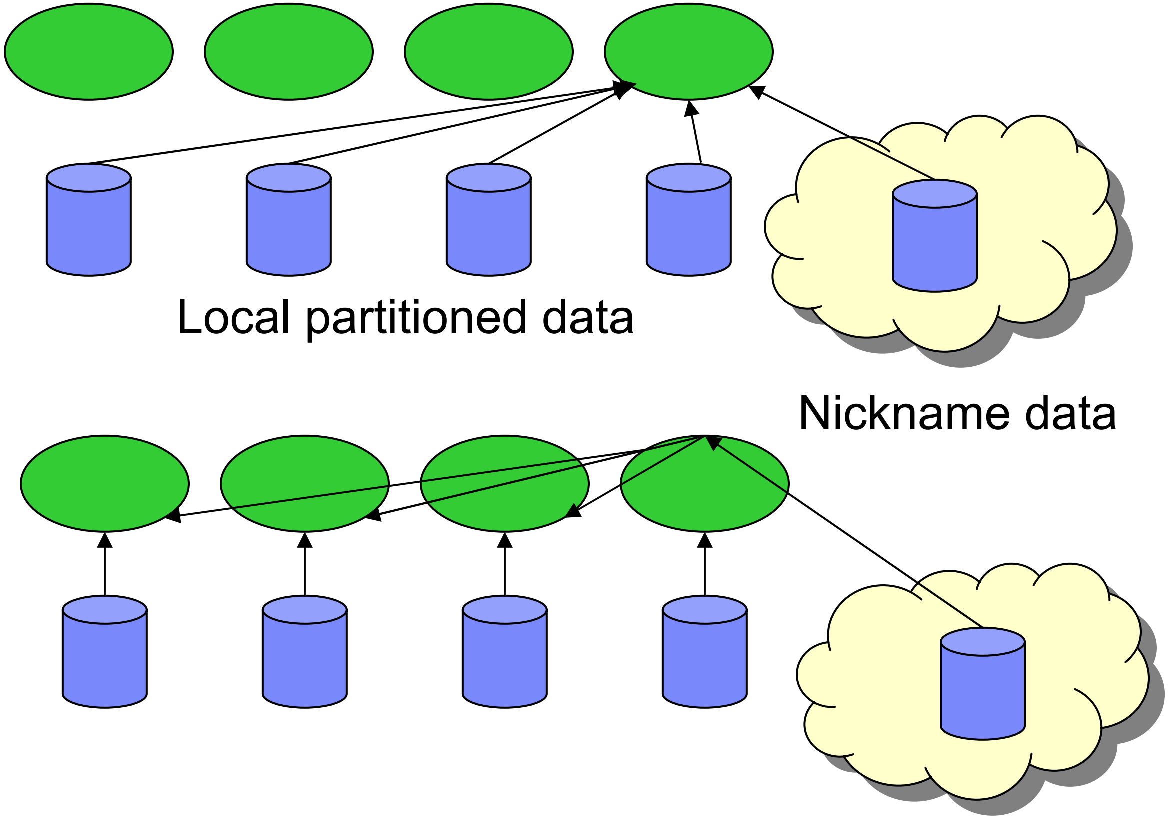


8 connections: 26% saved overall



32 connections: 63% saved overall

Better Integration of DB2 II into MPP systems: Joins [inserts, unions] of nicknames with local partitioned data

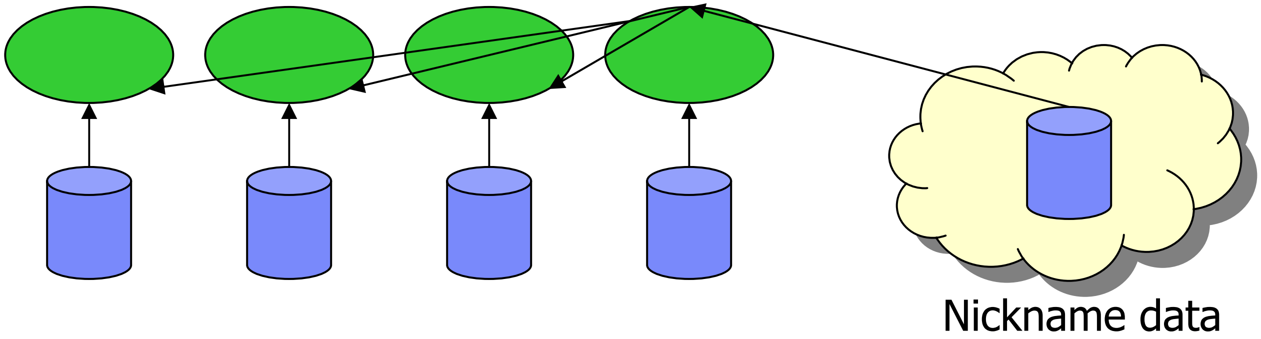


In DB2 II V8.1, nickname data and partitioned data can only be joined serially at the coordinator partition.

In DB2 II V8.2, nickname data can be distributed to all partitions. Joins to local partitioned data can execute in parallel. Requires fenced wrapper!

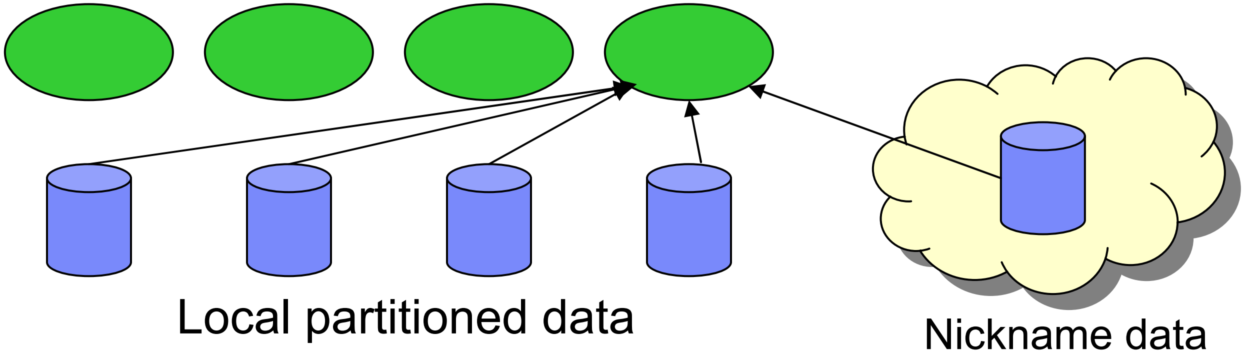
When are parallel plans involving local partitioned tables and nicknames helpful?

- Possible for joins or UNIONs between local and nickname data
- Parallel plans that redistribute nickname data to partitions of a local table or local join result make sense when
  - There is a large amount of local data involved in the query, especially if two or more local partitioned tables are involved
  - Why? Avoid moving a lot of local data to the coordinator
- Possible parallel plans
  - Broadcast or distribute nickname data to partitions of a local table or intermediate join result
  - Redistribute both nickname and local data to achieve a parallel join on a non-partitioning key



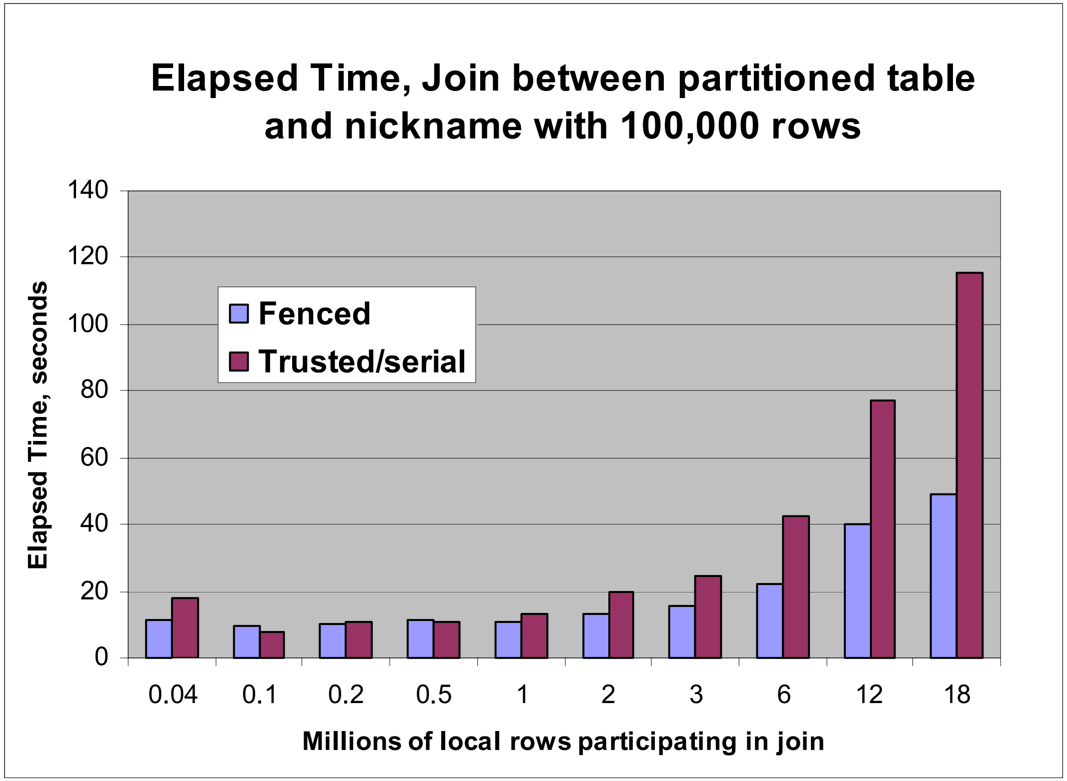
Will the optimizer always choose a parallel plan to join or UNION partitioned tables and nicknames?

- No. Serial plans at the coordinator are still best when
  - There is a lot of nickname data participating in the query
  - There is not so much local data participating in the query
  - Why? Doesn't make sense to redistribute a lot of nickname data if local data can quickly be moved to the coordinator



- Use of fenced wrapper enables, doesn't force parallel plan
- The optimizer worries about making the right choice!

Joining a partitioned table of varying size with a nickname of constant size



- Nickname with trusted wrapper: Serial plan **always** chosen
- Nickname with fenced wrapper: DB2 II **may** choose parallel plan
- Benefit of parallel plan is greatest if many rows from local table participate in join!
- How important is amount of nickname data?

MPP Plan with two local partitioned tables joined to a nickname. Fenced wrapper allows parallel plan that avoids merging local data at coordinator

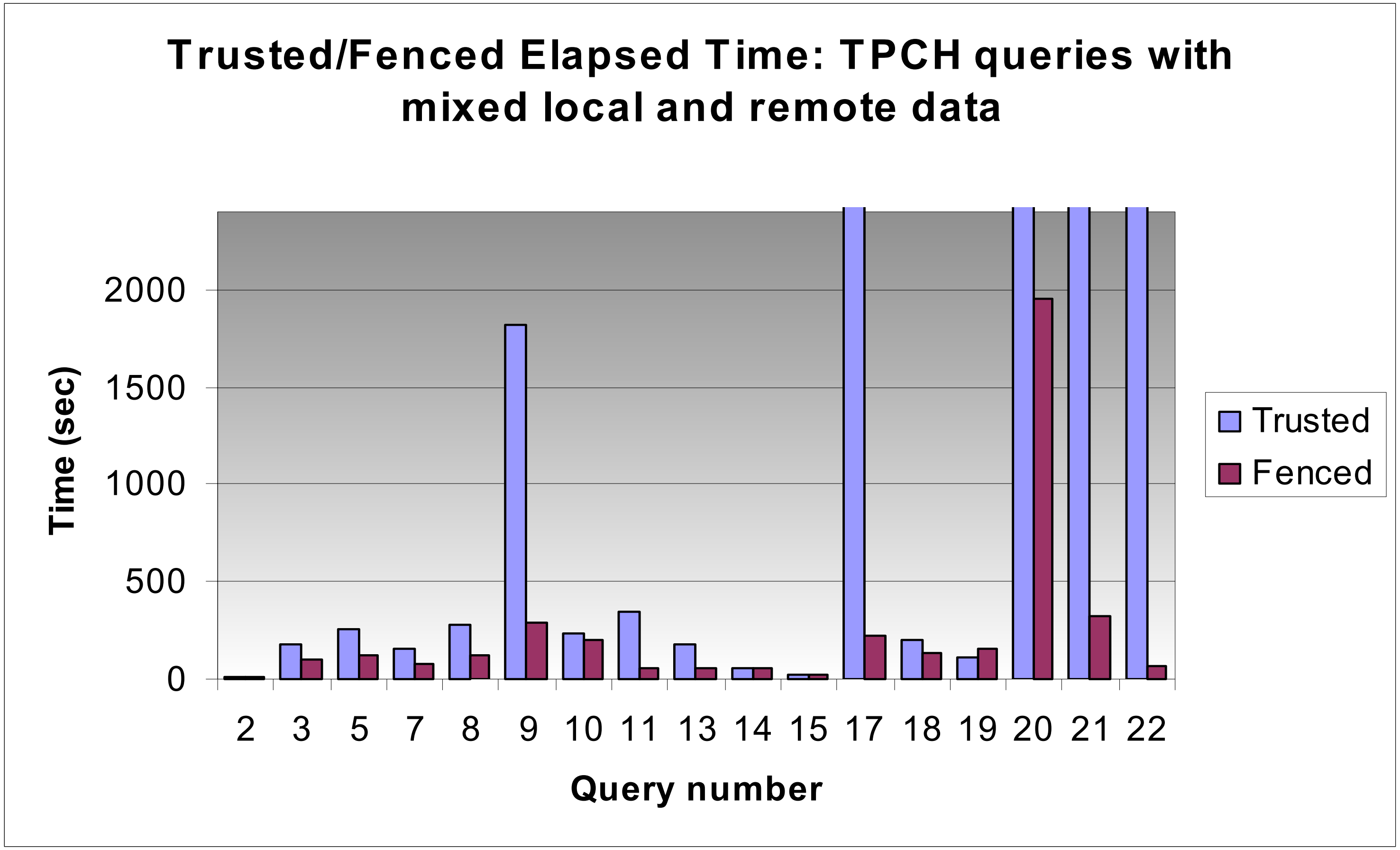
```
                RETURN
                  |
                  125
                GRPBY
                  |
                  125
                TBSCAN
                  |
                  125
                SORT
                  |
                966440
                HSJOIN
                  |
          /--------+--------\
     600000                    1.19935e+06
     SHIP                         HSJOIN
       |                            |
       |                 /-----+-----\
     600000           4.79698e+07      1.19935e+06
NICKNM: IFX4            DTQ              DTQ
    CUSTOMER             |                |
                  5.99622e+06        149919
                    TBSCAN           TBSCAN
                      |                |
                  5.99622e+06      1.49919e+06
                TABLE: TPCD      TABLE: TPCD
                   LINEITEM          ORDERS
```

Trusted: 738 sec

```
               RETURN
                  |
                 125
                DTQ
                  |
                15.625
                GRPBY
                  |
                  125
                MDTQ
                  |
                  125
                GRPBY
                  |
                  125
                TBSCAN
                  |
                  125
                SORT
                  |
                120805
                HSJOIN
                  |
          /--------+--------\
      75000                   149919
     DTQ                      DTQ
       |                        |
     600000                   149919
     SHIP                     HSJOIN
       |                        |
       |                 /-----+-----\
     600000       5.99622e+06        149919
NICKNM: IFX1        TBSCAN           TBSCAN
    CUSTOMER          |                |
                  5.99622e+06      1.49919e+06
                TABLE: TPCD      TABLE: TPCD
                   LINEITEM          ORDERS
```

Fenced: 161 sec

Effect of MPP parallelism for queries on a mix of local and remote data

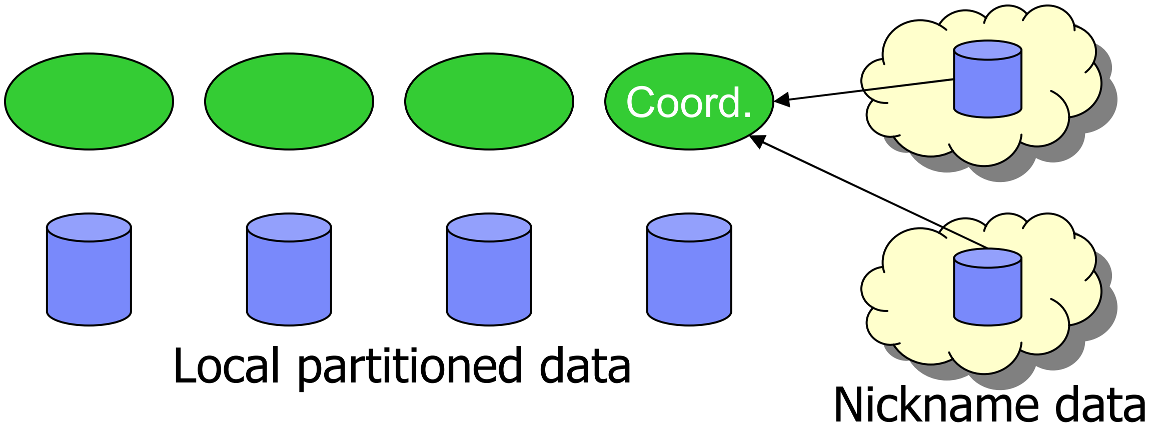


This workload: 10X improvement in overall elapsed time!

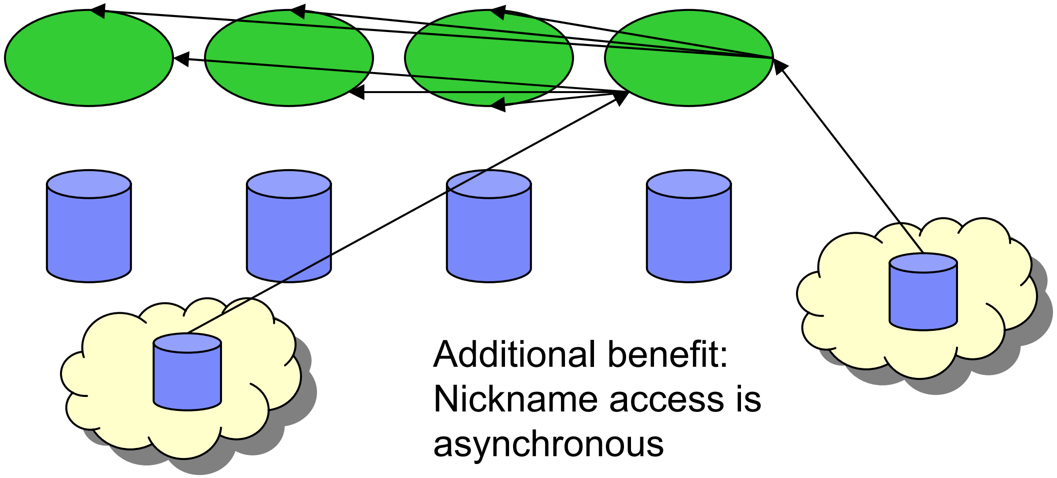
Benefit of parallel plans involving nicknames and local partitioned data

- Benefit is greatest when many local rows participate in the join
  - More than a few hundred thousand
- No benefit or even degradation if amount of local data involved in join is small
  - Best plan is still [serial] join-at-coordinator
  - Fenced wrapper overhead is noticeable
- Primary effect is avoiding moving local data to the coordinator
- This effect is amplified if multiple local tables participate in the join!
  - Pre-II 8.2: Joins of local tables had to be done at the coordinator before joining to a nickname.
  - II 8.2: Joins of multiple local tables and nicknames can be fully parallelized
- Moral: Almost always used fenced wrapper in for federated queries in MPP

Better Integration of DB2 II into MPP systems: Repartitioned parallel joins



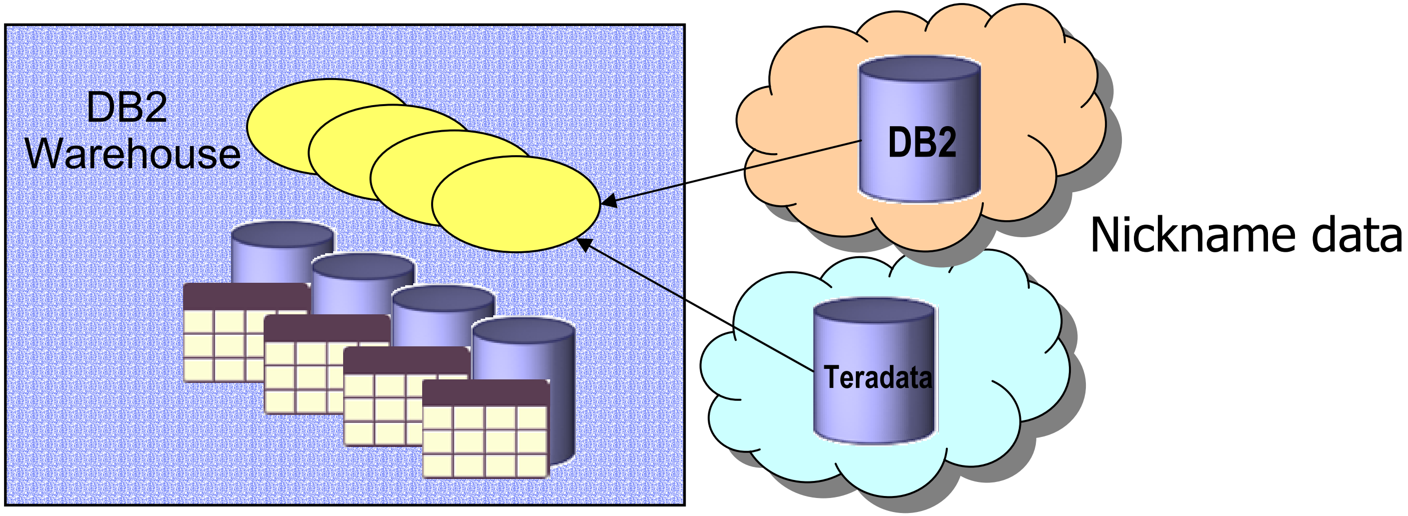
In DB2 II V8.1, two nicknames can only be joined serially at the coordinator partition.



Additional benefit: Nickname access is asynchronous

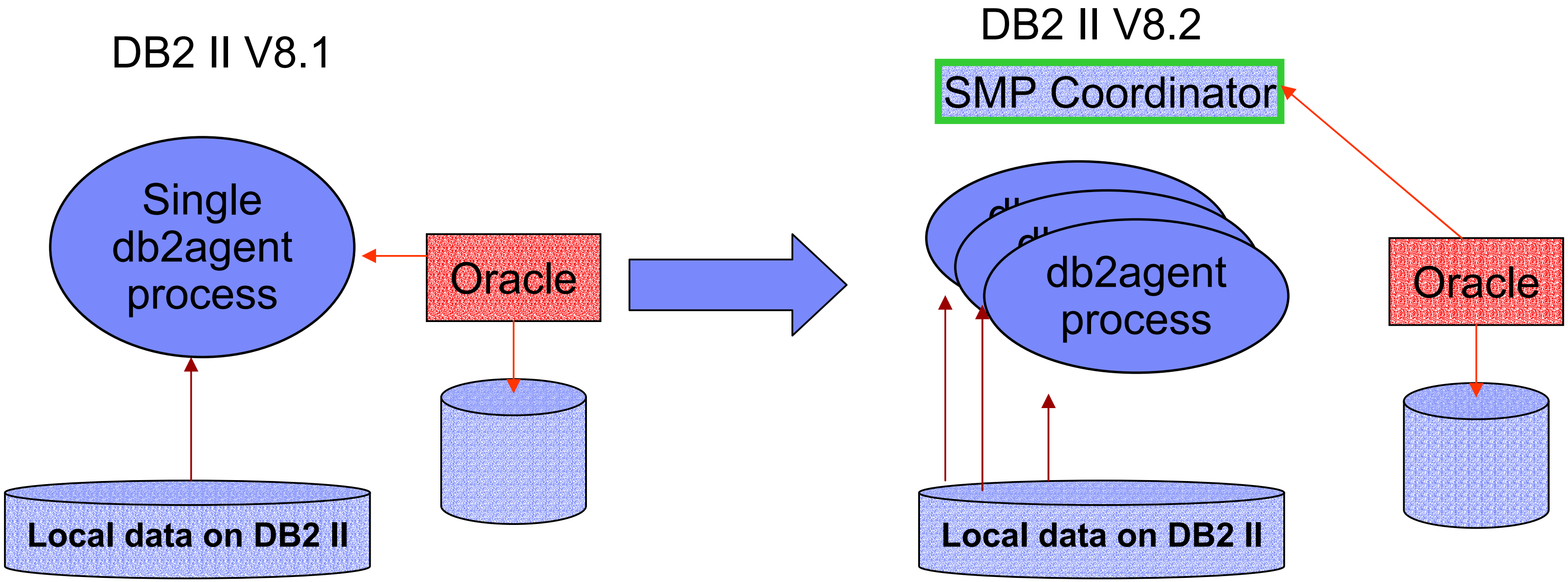
DB2 II V8.2 enables distribution of nickname data to a “computational partition group” for parallel joins.
Helpful for very large nickname-only joins

Extending a DB2 warehouse to access remote data using II: Placement of the II instance



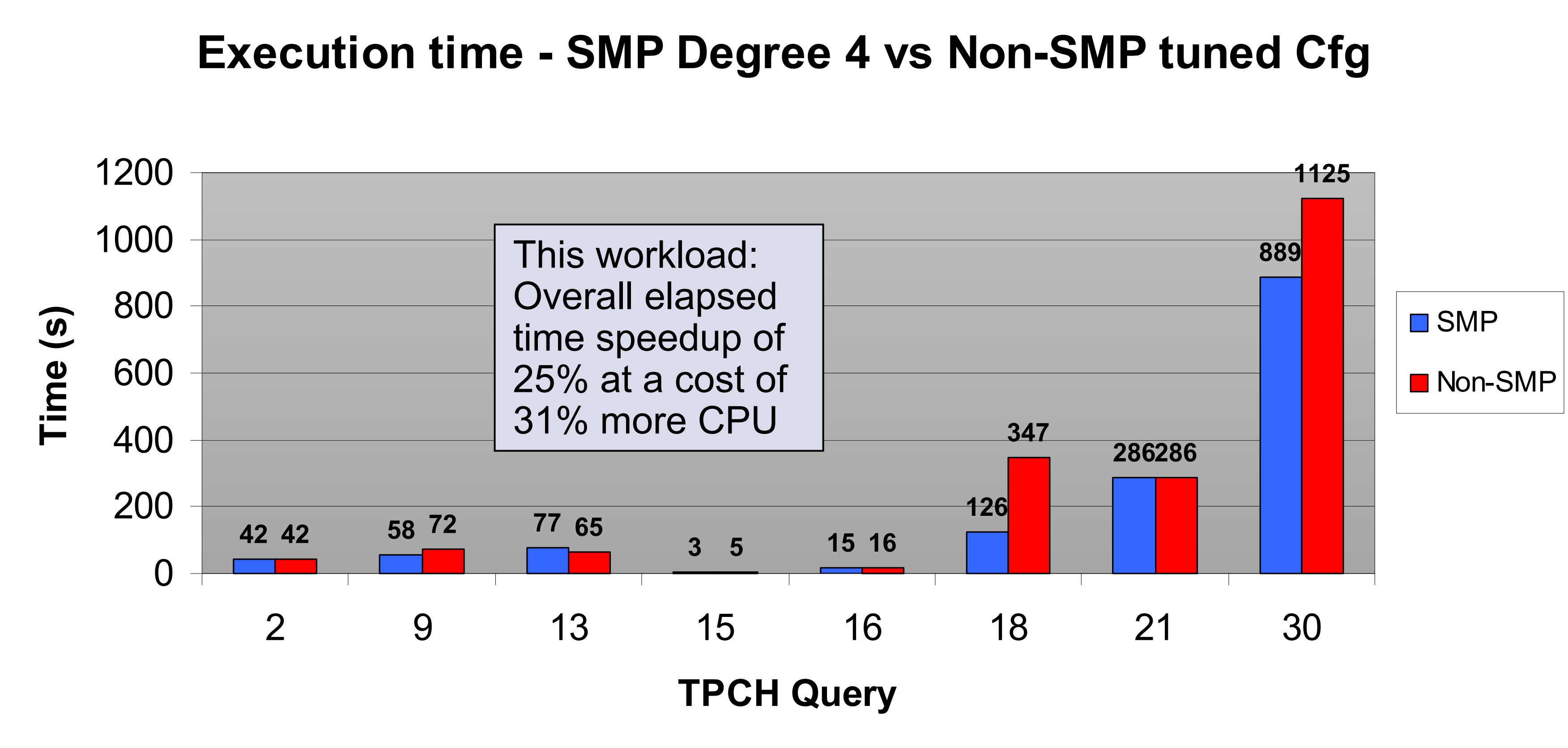
- Should the DB2 II instance be separate and "point" to both the warehouse and the remote data?
- Or should the DB2 warehouse itself be an II instance?
- The latter! Why?
  - Avoids movement of large amounts of data from warehouse to II instance
  - If the DB2 warehouse is partitioned: queries involving remote data can execute in parallel

Better integration of DB2 II into SMP systems



- Local portions of queries can now be executed in parallel
  - IF INTRA_PARALLEL = ‘YES’ and DEGREE > 1
- Does not depend on use of fenced wrapper
- Still no parallel access to remote data

Impact of SMP Parallelism for queries with mixed remote and local data



This workload: Overall elapsed time speedup of 25% at a cost of 31% more CPU

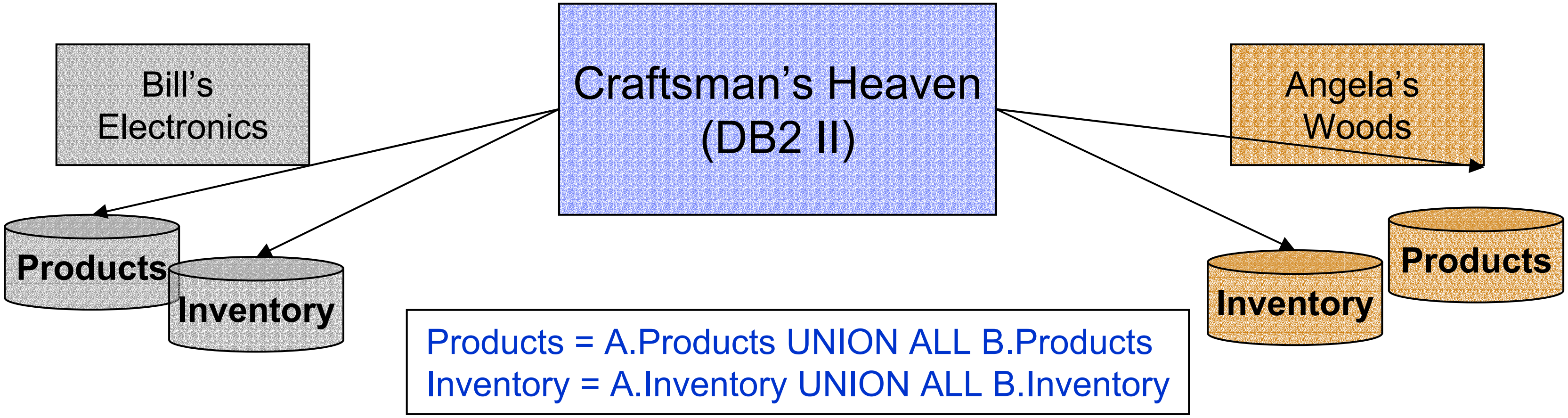
Enable using INTRA_PARALLEL "YES" and DFT_DEGREE > 1

When can SMP parallelism help federated queries?

- If the queries involve substantial processing of local data, in addition to federated access
- If INTRA_PARALLEL = ‘YES’ and DEGREE > 1
  - Advisable only if there are available CPU cycles
  - If system is already swamped, leave INTRA_PARALLEL = ‘NO’ (default)
- SMP-parallel processing of local data processing can improve response time at the cost of increased CPU consumption

Query Processing: UNION ALL View enhancements

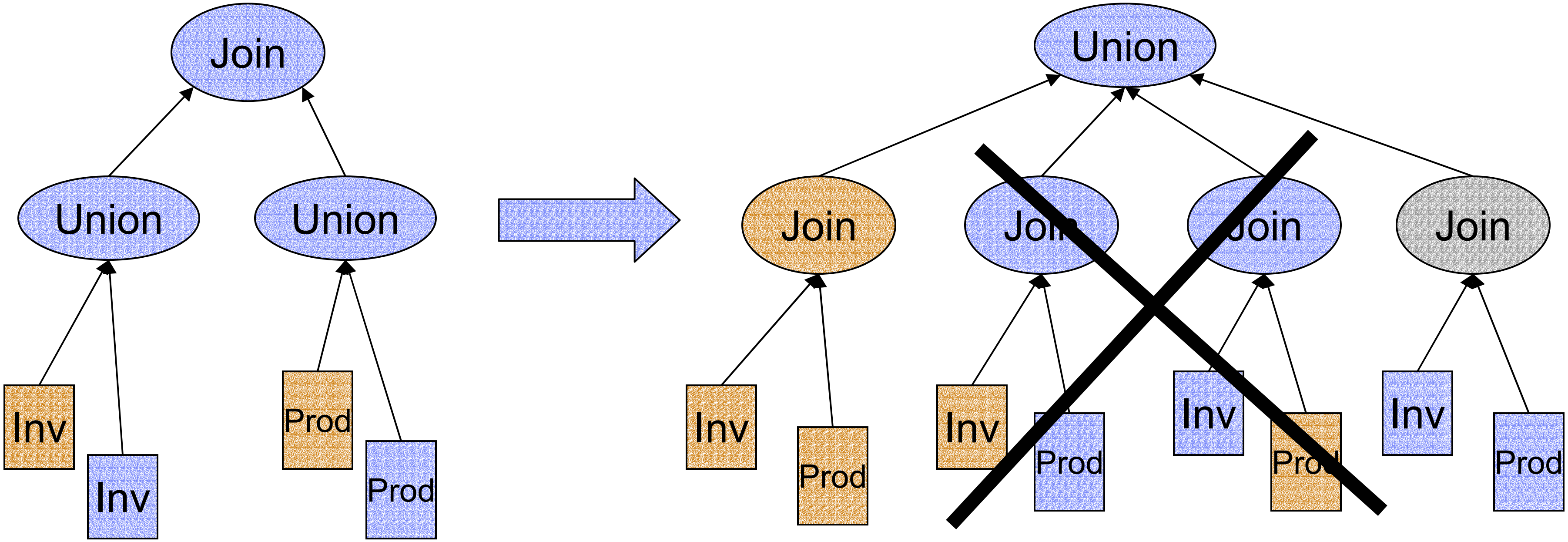
- UNION ALL views are an important federated modeling tool
  - Model mergers (both companies have client databases)
  - Model geographic distribution (each site tracks own inventory)
  - Model organizational structure (each division has own employees)
- An example: Merger of Angela’s Woods, Bill’s Electronics to form Craftsman’s Heaven
  - Create DB2 II nicknames to Angela’s and Bill’s Inventory and Product tables.
  - Create DB2 II UNION ALL views to merge Inventory and Products



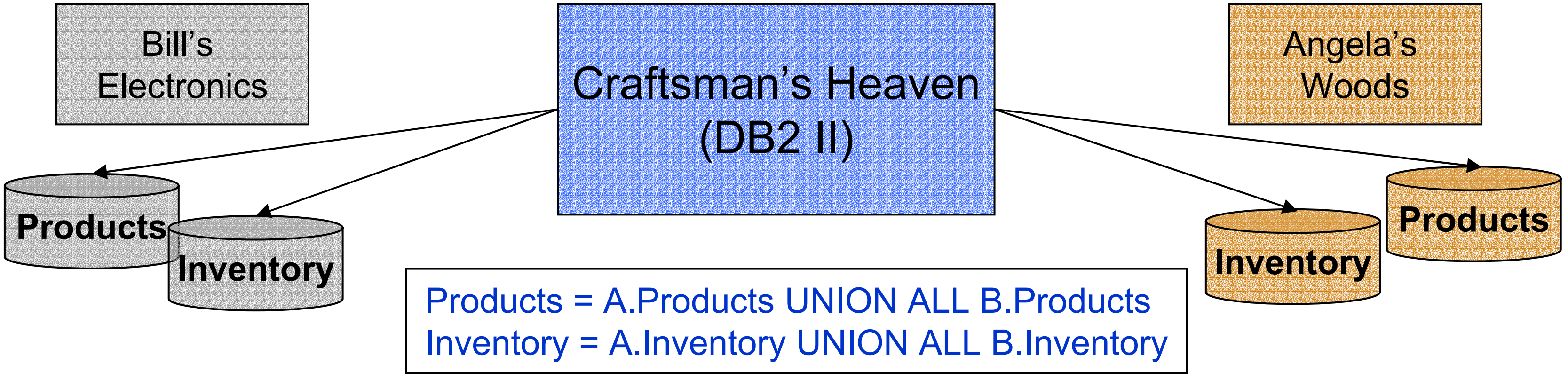
Products = A.Products UNION ALL B.Products
Inventory = A.Inventory UNION ALL B.Inventory

Using federated UNION ALL views

- Query processing challenge:
  - Join explosion when joining UNION ALLs. Why?
  - Join of UNIONs is same as UNION of join of all possible combinations of legs
- Example: Join Inventory and Products to find low inventory
  - Joins of Angela's products with Bill's inventory will yield nothing
  - Would be nice to tell the optimizer that to avoid these joins!



Using UNION ALL views over nicknames (detail)



Products = A.Products UNION ALL B.Products
Inventory = A.Inventory UNION ALL B.Inventory

1.) Create nicknames on DB2 II server:A.PRODUCTS, B.PRODUCTS, A.INVENTORY, B.INVENTORY

2.) Now create UNION ALL view for Products; similar for Inventory:

```
CREATE VIEW Products  AS
SELECT a.Products.*, 'Angela' AS store_id FROM a.Products
UNION ALL
SELECT b.Products.*, 'Bill' AS store_id FROM b.Products;
```

3.) To join Products and Inventory and tell optimizer to get rid of cross-store joins:

```
SELECT … FROM Products P, Inventory I WHERE … <join predicate between P and I>
AND P.store_id = I.store_id
```

4.) Result: Unproductive cross-store joins are pruned away. Query becomes a UNION ALL of pushed-down joins.

Craftsman's Heaven expands!

- Acquires 10 more stores with different crafts products; now we have UNION ALL views with 12 branches!
- Joining Inventory and Products produces 144 possible joins;
  - Only 12 of them are fruitful; rest must be "pruned" away
  - DB2 II V8.1 couldn't do this for large UNION ALL views
- No problem with DB2 II V8.2
  - New algorithm prunes unproductive joins during expansion phase
  - Handles unions up to 36 branches, Can join as many views as desired when
    - All UNION ALL views have matching branches with the same "partitioning" constraint (store id, here – Angela's vs. Bill's).
    - The query has an equijoin predicate on the "partitioning" column
- Further rewriting improvements enhance performance
  - Outer joins pushed through the unions
  - Richer set of predicates (including some subqueries) pushed through the unions

Informational constraints over nicknames

- Can now define informational constraints involving any combination of nicknames and tables
  - Relational integrity (R.I.)
  - Check constraints
  - Functional dependency
- “Informational” means “not enforced” But can be of great help to optimizer (esp. query rewrite component)
- Syntax analogous to that for tables. For example:

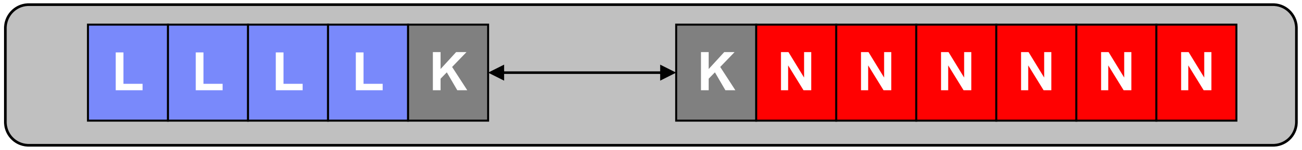
  alter nickname **lineitem** add constraint tryme

  foreign key (l_orderkey) references **orders**(o_orderkey) not enforced

  enable query optimization;

Informational constraints over nicknames: Sample usage scenarios:

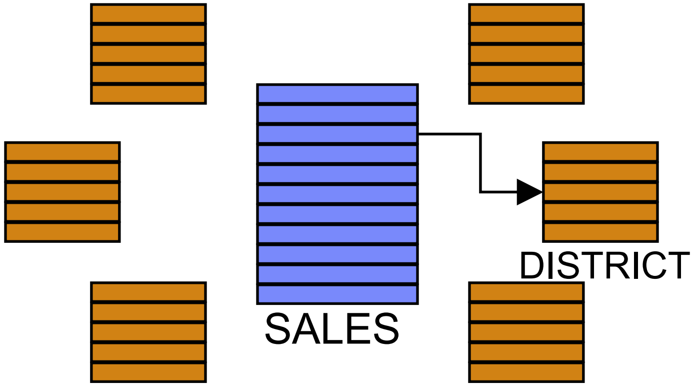
- Define a join view on DB2 II instance between local table and nickname using a “linking” key *K* and an R.I. constraint
- Define unique key K on the nickname (parent)
- Define foreign key K on the local table (child)
- For queries against the join view:
  - Can safely eliminate join to nickname if query needs only local columns



- Branch elimination in UNION ALL views over nicknames
  - Can add check constraints to nicknames to make branches disjunct

Informational constraints over nicknames: Benefits for MQT routing

- Example: fact-dimension joins on nicknames
  - Assume nicknames SALES (fact) and DISTRICT(dimension)
  - Suppose we have a local MQT over nicknames defined as:
    - “Select district_name, <sales_facts> from SALES, DISTRICT where SALES.district_id = DISTRICT.district_id group by district_name”
  - When can a query that wants “Total sales over all districts” use the MQT?
  - We can use the MQT if we are sure that every SALES.district_id has a matching DISTRICT.district_id
  - Must have R.I. Constraint between SALES and DISTRICT on district_id!



Improved ability to monitor federated queries

- Snapshot monitor can now display information about remote query fragments
- Steps:
  - UPDATE MONITOR SWITCHES USING STATEMENT ON;
  - GET SNAPSHOT FOR DYNAMIC SQL ON <DB_NAME>
- Output now shows
  - Federated statement as a whole, timing, rows returned, etc.
  - Remote statements pushed down to each individual source, with individual timings, rows returned
- Easier to debug long-running federated queries
  - Is the problem remote or local?

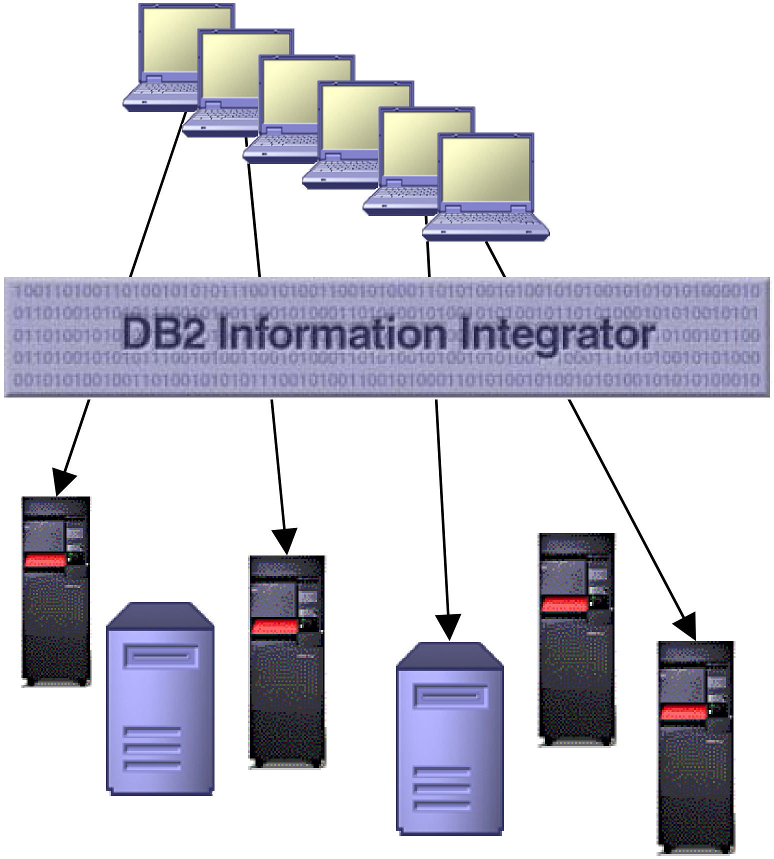
Agenda

- DB2 Information Integrator basics
  - Basic components, how it works, pushdown, what impacts performance?
- Federated queries against a single remote source
  - When is performance “good?” Reading an EXPLAIN. Diagnosing performance problems.
- Federated queries involving multiple sources
  - What does good performance mean? Useful comparisons
- Configuring DB2 II for best performance
  - Server options, type and function mappings. Statistics. MQTs.
- New features in II V8.2 that can improve performance
  - Parallel execution in SMP and MPP environments
  - Fenced wrapper, informational constraints, better monitoring
- Using DB2 II as a part of a solution
  - “Appropriate deployment” – when should you use it?
  - Know your workload! Complement DB2 II with caching/replication

Getting the best performance from DB2 II: Best practices in common usage scenarios

- Use as a “gateway” to a single remote system at a time
  - But careful… performance may not be what you expect
- Use as a multi-source data integration engine:
  - Know your queries. Don’t allow unregulated ad-hoc access.
- Use DB2 II in partnership with other techniques:
  - Caching of remote data locally via MQTs
  - Replication to enhance currency
  - Data consolidation where appropriate
  - Don’t use DB2 II to do “ETL”

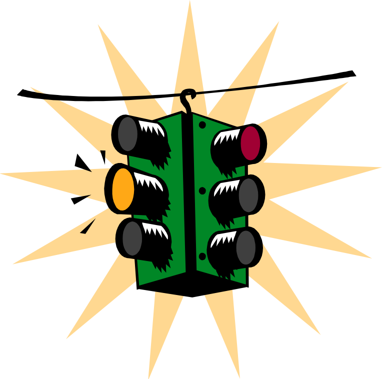
Best Practices: Using DB2 II as a "Gateway" to disparate remote databases



- Can use DB2 Information Integrator to unify application view of/access to different kinds of remote sources
- Primary access by this workload is to one source at a time
- May result in performance surprises!
  - Significantly slower than native access if result sets are large
  - Not all queries may be pushed down to relational sources as expected.
- Convenient, but not "free"
- Know your likely query patterns

Best Practices: Multi-source data integration

- Know your sources and remote objects.
  - How big are things? Tables? Views?
  - What indexes are there?
- Know your queries and their impact on remote sources
  - Performance of federated queries can vary widely depending on patterns of access
- Control your queries
  - Parameterized reports
  - Application controls
  - Goal: Limit access patterns to those with known performance characteristics
  - Consider Query Patroller



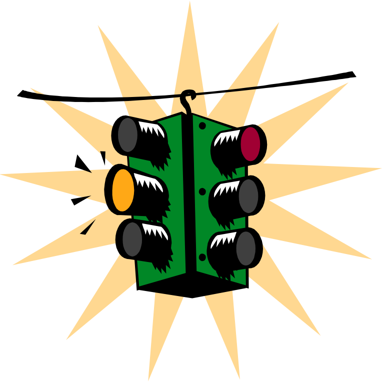
Best Practices: Multi-source data integration

- Know your typical queries
  - What local tables and nicknames do they access?
  - What are the likely execution plans?
  - Gather explain plans during proof-of-concept! (DB2EXFMT or Visual Explain) Be aware of:
    - Queries that move many rows to DB2 II
    - Nested joins with many probes to nickname inners
  - What remote queries will be generated? How will they impact your remote sources?

```
                 |
           4.79999e+06
              HSJOIN
              (   4)
          /------+-----\
   4.99999e+06         479999
     SHIP              SHIP
     (   5)            (   7)
       |                  |
      6e+06            600000
NICKNM: TPCD1     NICKNM: TPCD2
     ORDERS           CUSTOMER
```

```
             NLJOIN
             (   3)
           /----+---\
      185000         10
      TBSCAN        SHIP
      (   4)        (   9)
         |             |
     185000          6e+06
      SORT      NICKNM: ORA1
      (   5)        ORDERS
         |
     185000
      SHIP
      (   6)
         |
      600000
 NICKNM: ORA2
     CUSTOMER
```

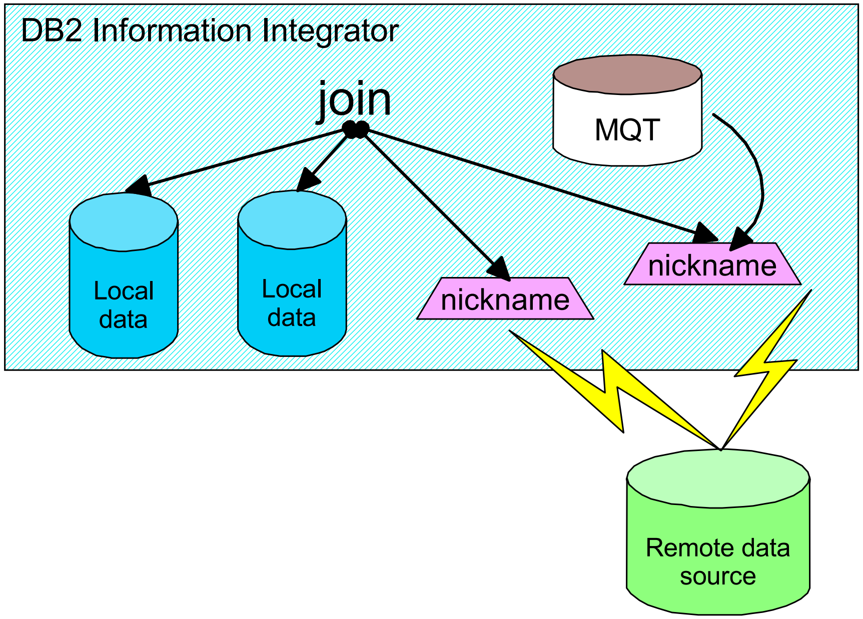
Best Practices: Multi-source data integration



- *Don't* allow unregulated ad-hoc federated access
- Consider Query Patroller
  - *Can* block queries from beginning execution if total estimated cost (at DB2 II server and all remote sources combined) exceeds a threshold.
  - *Can* limit concurrency at the II server
  - *Cannot* kill "runaway" queries.
  - *Cannot* be configured to treat one remote source differently from another
- Don't use DB2 II as an "ETL engine"
  - Not all transformations expressible as SQL
  - Use of complex transformation functions affect pushdown

Best Practices: Complement DB2 II federation with caching and replication

- DB2 II is seldom a solution on its own
- If possible, cache results of expensive remote queries locally at the DB2 II server using an MQT to enable reuse
- If needed often, consider caching part of a remote table locally (MQT). Keep in sync via replication if appropriate



Best Practices: Complement DB2 II federation with caching, replication and consolidation

- Federation in its place!
  - Use it to access current data in a targeted way
  - Not suitable for response-time critical queries that need to move large volumes of data in real time.
- Caching/replication helps by
  - Providing fast access to a local copy of near-current data
  - Re-using results of expensive queries that involve stable remote data
- If caching/replication is not practical
  - Take a hard look at consolidating some data on a permanent basis

Best Practices: Summary

- Federation: Control the kinds and cost of queries that are submitted to DB2 II using
  - Application controls: parameterized reports
  - DB2 Query Patroller
- Expect operations that move a lot of data between remote sources and DB2 II to take a long time
  - DB2 II used as a gateway works well, but can incur a significant performance cost
  - Don’t try to use DB2 II to implement a “virtual warehouse” on a permanent basis, especially if ad-hoc access to large data volumes is desired
  - Aim for “targeted” access to remote data
- Cache frequently-used data locally for best performance
  - Replicate for quick/high-volume access to near-real-time data
  - Federate for lower-volume slower access to real-time data