Query Transformation

Huong Tran
IBM

November 2021

Session 2AQ

GSE UK Conference 2021 Charity Raffle

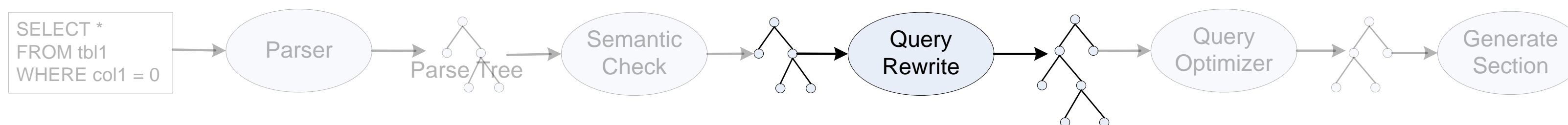
- The GSE UK Region team hope that you find this presentation and others that follow useful and help to expand your knowledge of z Systems.
- Please consider showing your appreciation by kindly donating to our charities this year, Royal National Lifeboat Institution (RNLI) & Guide Dogs for the Blind. Then follow the link on your receipt to enter your receipt number & amount donated into the GSE Raffle. You will get a raffle entry for every pound donated.
- Follow the link below or scan the QR Code:

http://uk.virginmoneygiving.com/GuideShareEuropeUKRegion

Supporting

Query Transformation

- After semantic processing, query transformation is the next stage in query processing.



Query Transformation

- **The way a query is written can force a certain access path to be chosen. Sometimes, there can be equivalent queries that generate the same results but provide more possible access path choices.**
- **The purpose of query transformation is to rewrite the query so that it opens up as many access path choices as possible.**
- **Transformation is useful when queries are not written with optimization in mind. This is especially true for queries that are automatically generated by applications, which are often complex and have many levels of nested subqueries.**

Query Transformation

- Query transformation rules can be categorized into:
  - Predicate Transformations
    - Predicate Pushdown
    - Transitive Closure
    - IN-List/BETWEEN to Equal
    - OR to IN-List
    - Predicate simplification
  - Join Transformations
    - Subquery to Join Transformation
    - Outer Join Transformations
  - View And Table Expression Transformations
    - View Merge
    - Table Expression Merge
  - UNION Transformations
    - Subselect Pruning
    - Distribution

Quick terminology review

- Predicate terms
  - **Simple or compound**
    - **A compound predicate is the result of two predicates.**
      - **Eg. WHERE (C1 = ? and C2 = ?)**
    - **All others are simple predicates**
      - **Eg. WHERE C1 = ?**
      - **In the COMPOUND example, the compound predicate is composed of two simple predicates connected by AND.**

Quick terminology review

- Predicate terms
  - **Local or join**
    - **Local predicates reference a single table**
      - **Eg. WHERE T1.C1 = ?**
      - **Also WHERE T1.C1 = T1.C2**
    - **Join predicates reference more than one table or correlated reference.**
      - **Eg. WHERE T1.C1 = T2.C1**
      - **Also WHERE SUBSTR(T1.C1,1,2) = T2.C1**

Quick terminology review

- Predicate terms
  - **Boolean term**
    - **Any predicate that is not contained in a compound OR condition**
    - **If boolean term is evaluated as FALSE for a particular row, the where clause is evaluated as FALSE for the row.**
    - **EG. Where (T1.C1 = ? AND T1.C2 = ?)**
  - **Non-boolean term examples**
    - **None of these predicates is considered "boolean term"**
    - **WHERE (T1.C1 = ? OR (T1.C2 = ? AND T1.C3 = ?))**

Predicate Transformations

- Predicate Transformations
    - Predicate Pushdown
    - Predicate Bubble-Up
    - Transitive Closure
    - IN-List/BETWEEN to Equal
    - OR to IN-List
    - Predicate simplification

Predicate Pushdown

- Push predicates into views and table expressions. If they must be materialized, then the database tries to create the smallest workfile possible.

```
SELECT    *
FROM
   (SELECT  REGION, YEAR, QTR, SUM(SALES)
    FROM     SALES_TABLE
    GROUP   BY REGION, YEAR, QTR)  QTR_SALES
WHERE QTR = 1
```

Predicate

```
SELECT    *
FROM
   (SELECT   REGION, YEAR, QTR, SUM(SALES)
    FROM      SALES_TABLE
    WHERE QTR = 1
    GROUP BY REGION, YEAR, QTR)  QTR_SALES
```

Push inside

Predicate Pushdown

- Pushdown supported predicate types
  - *column* OP *literal*
    - OP can be: =, <>, >, <, <=, >=
  - [NOT] LIKE
  - [NOT] BETWEEN *literal* and *literal*
  - *column* IS [NOT] NULL
  - IN-List

Predicate Pushdown

- Manually pushdown required for other predicates
  - **Predicate refers to materialized view / table expression**
    - **Compound predicates**
      - **WHERE (T1.C1 = ? OR T1.C2 = ?)**
    - **Join predicates**
    - **Subquery predicates dependent on materialization**
  - **Sometimes large inefficient materializations have predicates not pushed into table expression.**
  - **Consider manual pushdown to improve performance**

Predicate Pushdown

- Manual predicate pushdown:

```
SELECT    *
FROM
   (SELECT  REGION, YEAR, QTR, SUM(SALES)
    FROM      SALES_TABLE
    GROUP   BY REGION, YEAR, QTR)  QTR_SALES
WHERE (QTR = 1 OR YEAR < '1999')        <-- Predicate

SELECT    *
FROM
   (SELECT   REGION, YEAR, QTR, SUM(SALES)
    FROM       SALES_TABLE
    WHERE     (QTR = 1 OR YEAR < '1999')     <-- Manually Push inside
    GROUP BY REGION, YEAR, QTR)  QTR_SALES
```

Predicate Bubble-Up

- Local filtering within correlated subquery

```
SELECT T1.*
FROM T1                              ← No local filtering on outer table
WHERE EXISTS
        (SELECT 1
        FROM T2
        WHERE T1.SSN = T2.SSN         ← Correlation predicate
        AND  T2.SSN = '123-45-6789')  ← Local predicate on
                                        same column as
                                        correlation predicate
```

- In this example, subquery is dependent on table T1.

Predicate Bubble-Up

- Local filtering within correlated subquery

```
SELECT T1.*
FROM T1
WHERE EXISTS
        (SELECT 1
         FROM       T2
         WHERE      T1.SSN = T2.SSN
         AND  T2.SSN = '123-45-6789'
         AND  T1.SSN = '123-45-6789' )        ← (1) Replicate predicate
AND T1.SSN = '123-45-6789'                    ← (2) Pull predicate out
```

- Now T1 has local predicate on SSN.

Transitive Closure

Transitive closure provides more predicates for filtering, and possibly more join conditions for choosing join permutations.

- Before:

```
SELECT T0.C2, X.C2
FROM  T0, (SELECT DISTINCT T1.C1, T2.C2 FROM T1, T2
           WHERE T1.C3 > T2.C3) X
WHERE T0.C0 = X.C1 AND
      T0.C0 > 100;
```

- After:

```
SELECT T0.C2, X.C2
FROM  T0, (SELECT DISTINCT T1.C1, T2.C2 FROM T1, T2
           WHERE T1.C3 = T2.C3 AND
           T1.C1 > 100) X
WHERE T0.C0 = X.C1 AND
      T0.C0 > 100 AND
      X.C1 > 100;
```

Transitive Closure

- Supported local predicates
  - *column* OP *literal*
    - OP can be: =, <>, >, <, <=, >=
  - *column* [NOT] BETWEEN ? AND ?
  - IN-list

- PTC not supported for all predicates.
  - LIKE
  - *column* IN (*non-correlated subquery*)
  - compound predicates

Transitive Closure

- Predicate transitive closure can be applied to both local and join predicates:
  - Local predicates
    This allows for additional filtering on T2.

```
SELECT cols
FROM T1, T2
WHERE T1.C1 = T2.C1
AND T1.C1 = 30
AND T2.C1 = 30
```

  - Join predicates
    This creates an additional join predicate.

```
SELECT cols
FROM T1, T2, T3
WHERE T1.C1 = T2.C1
AND  T2.C1 = T3.C1
AND T1.C1 = T3.C1
```

Transitive Closure

- Transitive closure is not performed on all predicates.

```
SELECT cols
FROM T1, T2, T3
WHERE T1.C1 = T2.C1
AND T2.C1 = T3.C1
AND T1.C1 = T3.C1
AND T1.C1 IN (SELECT C1 FROM T4 WHERE T4.Cx = ?)
AND T2.C1 IN (SELECT C1 FROM T4 WHERE T4.Cx = ?)
AND T3.C1 IN (SELECT C1 FROM T4 WHERE T4.Cx = ?)
AND T1.C1 IN ('A', 'B', 'C')
AND T2.C1 IN ('A', 'B', 'C')
AND T3.C1 IN ('A', 'B', 'C')
AND T1.C1 LIKE 'XX%'
AND (T1.C1 = ? OR T1.C2 = ?)     <-- only uses join predicates…
```

- We could add generate three additional predicates for T3 based on the three T1 predicates in blue, but this is not done. This is because those predicates cannot lead to a faster access path.

Transitive Closure

- Why these predicates not transformed
  - **LIKE predicate can eliminate previous index only access**
  - **IN-list / compound predicates**
    - **Prepare cost to look in compound cases expensive**
    - **Can cause more SQLCODE -101 errors**
  - ***column* IN SUBQUERY**
    - **SUBQUERY would be executed twice**

  - **More research required to extend transitive closure while avoiding pitfalls.**

  - **Recommendations:**
    - **Consider manually adding these predicates**
    - **Will increase optimization opportunities**
    - **New opportunity may be more efficient access path**

OR to IN-List

- OR to IN-List:
  - C1 = 1 OR C1 = 2 OR C1 = 3 is rewritten as C1 IN (1, 2, 3)

- This transformation is performed because:
  - The column becomes a candidate for single-column in-list access.

IN-List/BETWEEN to Equal

- IN-List to Equal:
  - C1 IN (1) is rewritten as C1 = 1
- BETWEEN to Equal:
  - C1 BETWEEN 1 AND 1 is rewritten as C1 = 1

- These transformations are performed because:
  - IN and BETWEEN are not candidates for the predicate transitive closure transformation.
  - More statistics are usable for the equals predicate.
  - Between/IN-list can stop matching in index earlier

Predicate simplification

A series of rewrite are done for predicates containing YEAR(col), DATE (col), and SUBSTR(col,1,len) built-in functions to achieve a better performance.

- ‘YEAR(col) op value’ can have YEAR() removed
  - Col is Date, Timestamp, or Timestamp with time zone
  - Value is Integer value
- ‘DATE(col) op value’ can have DATE() removed
  - Col is Timestamp or Timestamp with time zone
  - Value is Date, Char, or Varchar
- ‘SUBSTR(col) op value’ can have SUBSTR() removed
  - Col is Char, Varchar, Graphic, or Vargraphic
  - Value has same data type and ccsid as column
- Op can be =, >,>=,<,<=
- Values can be constants, variables, or any invariant expressions
  - Invariant expressions do not change during statement evaluation

Simplify YEAR(col), DATE(col), and SUBSTR(col,1,len) Predicates

- Example of Invariant Expressions
  - 123
  - 123+456
  - Current Date – 7 Days
  - :first_name||' '||:last_name
- Example of Variant Expression
  - Column1
  - INTEGER(Column1)
  - RAND()
  - User-defined function with external action, is non-deterministic, or modifies SQL data

Simplify YEAR(col), DATE(col), and SUBSTR(col,1,len) Predicates

- Examples
  - SELECT * FROM t1 WHERE YEAR(col) = 2012;
    - If col is Date and IOE exists on YEAR(col), then
      - SELECT * FROM t1 WHERE YEAR(col) = 2012 AND col BETWEEN ‘2012-01-01’ AND ‘2012-12-31’;
    - If col is Timestamp, then
      - SELECT * FROM t1 WHERE col BETWEEN ‘2012-01-01-00.00.00.000000’ AND ‘2012-12-31-24.00.00.000000’;
  - SELECT * FROM t1 WHERE DATE(col) = ‘2011-06-01’;
    - If col is Timestamp and IOE exists on DATE(col), then
      - SELECT * FROM t1 WHERE DATE(col) = ‘2011-06-01’ AND col BETWEEN ‘2011-06-01-00.00.00.000000’ AND ‘2011-06-01-24.00.00.000000’;

Simplify YEAR(col), DATE(col), and SUBSTR(col,1,len) Predicates

- Examples
  - SELECT * FROM t1 WHERE SUBSTR(col,1,3) = ‘San’;
    - If col is varchar(10) unicode, then
      - SELECT * FROM t1 WHERE col BETWEEN X’53616E00000000000000’ AND X’53616EFFFFFFFFFFFFFF’;

Simplify YEAR(col), DATE(col), and SUBSTR(col,1,len) Predicates

- Different op produce different rewrites
  - SELECT * FROM t1 WHERE YEAR(col) > 2011;
    - If col is Date then,
      - SELECT * FROM t1 WHERE col > ‘2011-12-31’;
  - SELECT * FROM T1 WHERE YEAR(col) >= 2011;
    - If col is Date then,
      - SELECT * FROM t1 WHERE col >= ‘2011-01-01’;
  - SELECT * FROM T1 WHERE YEAR(col) BETWEEN 2000 AND 2012;
    - If col is Date then,
      - SELECT * FROM T1 WHERE col BETWEEN ‘2000-01-01’ AND ‘2012-12-31’;

Simplify YEAR(col), DATE(col), and SUBSTR(col,1,len) Predicates

- Different op produce different rewrites…
  - SELECT * FROM t1 WHERE YEAR(col) IN (2011, 2012);
    - If col is Date then,
      - SELECT * FROM t1 WHERE (col BETWEEN ‘2011-01-01 AND ‘2011-12-31’ OR col BETWEEN ‘2012-01-01’ AND ‘2012-12-31’);

Simplify YEAR(col), DATE(col), and SUBSTR(col,1,len) Predicates

- Note: Predicate does not have to be a boolean term. Boolean term – if false, then the row does not qualify.
- If Index On Expression exists on the LHS expression, then the original predicate is left in the query
- Otherwise if no Index On Expression, then the original predicate is removed

Simplify YEAR(col), DATE(col), and SUBSTR(col,1,len) Predicates

- When is transformation not done:
  - HAVING predicate
  - CASE when-clause predicate
  - ON predicate if column is from the preserved side of the outer join.
  - LIKE predicate
  - Value is a variant expression

Join Transformations

- Join Transformations
  - Subquery to Join Transformation
  - Outer Join Transformations

Subquery to Join Transformation

- Subqueries are another opportunity where multiple SELECT blocks can be merged into a single SELECT block.
- A query containing a subquery has an implied execution sequence. If the it can be rewritten as a join, additional join sequences and methods can be considered.
- The subquery to join transformation may be applied to both correlated and non-correlated subqueries.

Subquery to Join Transformation

- Transform non-correlated subquery using IN predicate to a join.
- Before:

```
SELECT *
FROM  EMP
WHERE DEPTNO IN
         (SELECT DEPTNO FROM DEPT
          WHERE LOCATION IN ('SJ', 'SF')
                AND DIVISION = 'MARKETING')
```

- After:

```
SELECT EMP.*
FROM   EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
  AND DEPT.LOCATION IN ('SJ', 'SF')
  AND DEPT.DIVISION = 'MARKETING`
```

- DEPTNO values must be unique.

Subquery to Join Transformation

- Transform correlated subquery using IN predicate to a join.
- Before:

```
SELECT T1.C1
FROM   T1
WHERE  T1.C1 IN (SELECT T2.C1
                 FROM   T2
                 WHERE  T2.C2 = T1.C2)
```

- After:

```
SELECT T1.C1
FROM   T1, T2
WHERE  T1.C1 = T2.C1
       AND T2.C2 = T1.C2
```

Subquery to Join Transformation

- There are certain conditions that must be met in order to consider the subquery to join transformation.
  - The set comparison predicate can be: IN, = ANY, = SOME, EXISTS.
  - No column functions in subquery.
  - Single table in subquery.
  - The subquery does not contain any GROUP BY or HAVING clauses.
  - For non-correlated subqueries, there cannot be any duplicates returned from the subquery. This is due to the semantics of the IN keyword, in which duplicates in the subquery do not affect the results. When transforming the subquery into a join, duplicates in the subquery will create unwanted results.
  - This list is not complete. See the DB2 Administration Guide, section titled “Conditions for DB2 to transform a subquery into a join” for more details.

Correlated Subquery Considerations

- How can you influence when a correlated subquery will be evaluated?
  - Correlated subquery evaluated when all dependent tables are in the composite.
  - Often subqueries can be coded against any of a set of tables, or a whole set of tables
  - How you code the subquery will influence when it is evaluated.

Correlated Subquery Dependencies

- Presence of join predicates in parent select which are also used as correlation predicates on correlated subquery
  - Subquery could be written to be dependent on either T1, T2, or both

```
SELECT T1.*
FROM T1, T2
WHERE T1.C1 = T2.C1
AND T1.C2 = T2.C2             <-- Join predicates
AND T1.C2 =                            <-- SUBQ dependent on T1
         (SELECT MAX(T3.C2)
          FROM T3
          WHERE T1.C1 = T3.C1)         <-- SUBQ dependent on T1
```

- In this example, subquery is dependent on table T1.

Correlated Subquery Dependencies

- Presence of join predicates in parent select which are also used as correlation predicates on correlated subquery
  - Subquery could be written to be dependent on either T1, T2, or both

```
SELECT T1.*
FROM T1, T2
WHERE T1.C1 = T2.C1
AND T1.C2 = T2.C2               <-- Join predicates
AND T1.C2 =                              <-- SUBQ dependent on T1
         (SELECT MAX(T3.C2)
          FROM T3
          WHERE T1.C1 = T3.C1)       <-- SUBQ dependent on T1
```

- In this example, subquery is dependent on table T1.

Correlated Subquery Alternatives

Subquery dependent on T2

```
SELECT T1.*
FROM T1, T2
WHERE T1.C1 = T2.C1
AND T1.C2 = T2.C2
AND T2.C2 =                 <-- SUBQ dependent on T2
     (SELECT MAX(T3.C2)
      FROM T3
      WHERE T2.C1 = T3.C1)         <-- SUBQ dependent on T2
```

Subquery dependent on both

```
SELECT T1.*
FROM T1, T2
WHERE T1.C1 = T2.C1
AND T1.C2 = T2.C2
AND T2.C2 =                 <-- SUBQ dependent on T2
     (SELECT MAX(T3.C2)
      FROM T3
      WHERE T1.C1 = T3.C1)         <-- SUBQ also dependent on T1
```

Correlated Subquery Suggestions

- What difference does it make?
  - Subquery predicate cheap, highly selective?
    - Goal: Evaluate subquery early
    - Code against likely outer table
  - Subquery predicate expensive, low selectivity?
    - Goal: Evaluate subquery as few times as possible
    - Code against more filtered table
    - If significant selectivity after join of two tables, make dependent on both tables
  - Correlated subquery filtering and cost considerations:
    - High filtering estimate can move a table up in join sequence
    - Very expensive correlated subquery can push table to be joined later

Outer Join Transformations

- When possible, the database tries to transform:
  - full outer join to left outer join
  - left or right outer join to inner join
- This is desirable because a full outer join is implemented as a sort merge join, which could potentially be an expensive operation. Rewriting the query as a left outer join adds the possibility of using a nested loop join implementation.
- Transforming a left or right outer join to a inner join allows for more join sequences and methods that the optimizer can consider. It can also reduce materialization.

Outer Join Transformations

Full outer join to left outer join example.

- Before:

```
SELECT *
FROM T1 FULL OUTER JOIN T2 ON T1.C1 = T2.C1
WHERE T1.C2 > 100
```

- After:

```
SELECT *
FROM T1 LEFT OUTER JOIN T2 ON T1.C1 = T2.C1
WHERE T1.C2 > 100
```

- Any T1 null-padded rows will be removed by the WHERE clause, so FULL OUTER JOIN can be transformed into LEFT OUTER JOIN.

Outer Join Transformations

Left outer join to inner join transformation example.

- Before:

```
SELECT *
FROM T1 LEFT OUTER JOIN T2 ON T1.C1 = T2.C1
WHERE T2.C2 > 100
```

- After:

```
SELECT *
FROM T1 INNER JOIN T2 ON T1.C1 = T2.C1
WHERE T2.C2 > 100
```

or:

```
SELECT *
FROM T1, T2
WHERE T1.C1 = T2.C1
  AND T2.C2 > 100
```

- Any T2 null-padded rows will be removed by the WHERE clause, so LEFT OUTER JOIN can be transformed into INNER JOIN.

View And Table Expression Transformations

- View And Table Expression Transformations
  - View Merge
  - Table Expression Merge

View Merge

- When a view is created like:

```
CREATE VIEW V1 AS
SELECT T1.C1, T2.C2 FROM T1, T2 WHERE T1.C1 = T2.C1
```

- The database tries to avoid materializing the view. Materializing the view means that the database must execute the SELECT within the view definition and store the results before being able to select from the view.
- Instead, the database attempts to merge the view definition into a single SELECT statement. For example, the query:

```
SELECT V1.* FROM V1, T3
WHERE V1.C2 = T3.C2
  AND T3.C4 = ?
```

can be rewritten as:

```
SELECT T1.C1, T2.C2
FROM T1, T2, T3
WHERE T1.C1 = T2.C1
  AND T2.C2 = T3.C2
  AND T3.C4 = ?
```

If view V1 materializes
- T1 and T2 would be joined unfiltered.
- Possible large workfile
- No index access on workfile
- Fewer join sequences, join methods

With view merge we have more options
- Any join sequence
- More indexes available
- More join methods possible

Materialization

- Materialization is undesirable:
  - Optimization options are severely restricted:
    - Fewer join sequences are considered, so the chosen sequence may not be optimal.
    - The statistics on the materialized view will not be as accurate as a base table. This will cause the optimizer to make less accurate access path selection decisions.
    - There are no indexes on materialized views, so there are fewer efficient access methods.
    - Predicates are not pushed down into materializations.
  - The costs to materialize a view are considerable.
    - The entire view must be materialized before being able to retrieve records from it.
    - A materialization involves the base tables being read in, processed, and written back to disk.
    - Materialization can delay finding first row - user waits longer for fetches to start.

Merge Table Expressions

- Merging table expressions is similar:
- Before:

```
SELECT DEPT.DEPTNO, SUM(X.EDUC), COUNT(X.EDUC), SUM(X.EDUC)
       FROM DEPT, (SELECT EMP.EDUC, EMP.DEPTNO
                   FROM EMP
                   WHERE EMP.EMPNO < '500000') X(EDUC, DEPTNO)
       WHERE DEPT.NAME > 'KIH' AND DEPT.DEPTNO = X.DEPTNO
       GROUP BY DEPT.DEPTNO
```

- After:

```
SELECT DEPT.DEPTNO, SUM(EMP.EDUC), COUNT(EMP.EDUC), SUM(EMP.EDUC)
       FROM DEPT, EMP
       WHERE DEPT.NAME > 'KIH' AND DEPT.DEPTNO = EMP.DEPTNO
                AND EMP.EMPNO < '500000'
       GROUP BY DEPT.DEPTNO
```

Summary

- View and table expressions merge:
  - Allows for more join sequences
  - Eliminates expensive materialization
  - Avoids predicate pushdown limitations
  - Allows usage of indexes on base tables
    - Join predicates not pushed into materializations
    - Predicates not pushed down would not be indexable
- Suggestions:
  - Avoid views and table expressions that use DISTINCT and GROUP BY. These must be materialized.
    - If possible, pull DISTINCT and GROUP BY out of view into the SELECT (can be difficult)
  - Push as much filtering into view and table expressions as possible.
  - Rewrite non correlated table expression/view into correlated table expression/view.

Table Expression

- Rewriting a non-correlated table expression to a correlated table expression.

```
SELECT    * FROM
          EMP E
          , (SELECT DEPTNO,
  COUNT(*)
          FROM DEPT D
          GROUP BY DEPTNO) Dx
WHERE Dx.DEPTNO = E.DEPTNO
AND E.EMP_AREA = 'HAWAII';
```

```
SELECT    *  FROM      EMP E
          , TABLE
          (SELECT DEPTNO, COUNT(*)
          FROM DEPT D
          WHERE D.DEPTNO = E.DEPTNO
          GROUP BY DEPTNO) Dx
WHERE Dx.DEPTNO = E.DEPTNO
AND E.EMP_AREA = 'HAWAII';
```

- Table expresion Dx is a FULL AGGREGATION of DEPT table.
  - EMP table is substantially reduced with local predicate E.EMP_AREA = 'HAWAII'
  - Materialization is most expensive operation in query
- Correlate EMP and DEPT table by adding 'TABLE' clause, and correlation predicate
  - Major improvement ONLY WHEN
    - Correlation predicate provides much more efficient access to inner table
    - Outer table (EMP) is highly selective (few accesses to DEPT required)

UNION Transformations

- UNION Transformations
  - Subselect Pruning
  - Distribution

Subselect Pruning

- Avoid evaluating nested SELECT blocks that return no records.
- This transformation is most easily exercised for SELECT blocks linked together with UNIONs.
  - However, this technique can also be applied to prune non-UNION expressions.
- Start by comparing the predicates in the nested blocks with its parent blocks in the parse tree. When a contradictory predicate is encountered, we know not to evaluate that block.

Subselect Pruning

```
CREATE VIEW transaction( ...... ) AS
SELECT *
FROM first_season
WHERE date BETWEEN '2002-01-01' AND '2002-3-31'
  UNION ALL
SELECT *
FROM second_season
WHERE date BETWEEN '2002-04-01' AND '2002-6-30'
  UNION ALL
SELECT *
FROM third_season
WHERE date BETWEEN '2002-07-01' AND '2002-9-30'
  UNION ALL
SELECT *
FROM fourth_season
WHERE date BETWEEN '2002-10-01' AND '2002-126-31' ;
```

```
SELECT *
FROM transaction T, products P,
        customers C, dates D
WHERE T.pid = P.id    AND
        T.cid = C.id    AND
        T.did = D.id    AND
        C.zipcode IN ( ......)    AND
        T.date BETWEEN '2002-4-15'
                AND '2002-5-15' ;
```

Union Legs in the view are pruned before composing so we can reduce the view complexity and avoid SQLCODE -101

Distribution

- Distribute predicates, joins, and aggregates into nested SELECT blocks.
- Original query:

```
SELECT DEPT.DEPTNO, AVG(X.EDUC)
FROM DEPT, (SELECT EMP.EDUC, EMP.DEPTNO
            FROM EMP  WHERE EMP.EMPNO < '2000'
            UNION ALL
            SELECT EMP.EDUC,  EMP.DEPTNO
            FROM EMP WHERE EMP.EMPNO >= '5000' )
            X(EDUC, DEPTNO)
WHERE DEPT.NAME> 'KIH' AND DEPT.DEPTNO = X.DEPTNO
GROUP BY DEPT.DEPTNO
HAVING SUM(X.EDUC) > 30
```

Distribution

- Query after distribution:

```
SELECT Y.DEPTNO, CASE WHEN SUM(Y.COL03) = 0 THEN NULL
       ELSE SUM(Y.COL02)/SUM(Y.COL03) END
FROM ( SELECT DEPT.DEPTNO, SUM(X.EDUC), COUNT(X.EDUC), SUM(X.EDUC)
       FROM DEPT, (SELECT EMP.EDUC, EMP.DEPTNO
                   FROM EMP
                   WHERE EMP.EMPNO < '500000') X(EDUC, DEPTNO)
       WHERE DEPT.NAME > 'KIH' AND DEPT.DEPTNO = X.DEPTNO
       GROUP BY DEPT.DEPTNO
       UNION ALL
       SELECT DEPT.DEPTNO, SUM(X.EDUC), COUNT(X.EDUC), SUM(X.EDUC)
       FROM DEPT, (SELECT EMP.EDUC, EMP.DEPTNO
                   FROM   EMP
                   WHERE EMP.EMPNO >='500000') X(EDUC, DEPTNO)
       WHERE DEPT.NAME > 'KIH' AND DEPT.DEPTNO = X.DEPTNO
       GROUP BY DEPT.DEPTNO
     ) Y(DEPTNO, COL02, COL03, COL04)
GROUP BY Y.DEPTNO
HAVING SUM(Y.COL04) > 30
```

Summary

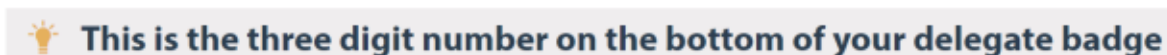
- Query transformation is used to rewrite the query so that it opens up as many access path choices as possible.
- The transformation rules that we covered:
  - Predicate Transformations
    - Predicate Pushdown
    - Predicate Bubble-Up
    - Transitive Closure
    - IN-List/BETWEEN to Equal
    - OR to IN-List
    - Predicate simplification
  - Join Transformations
    - Subquery to Join Transformation
    - Outer Join Transformations
  - View And Table Expression Transformations
    - View Merge
    - Table Expression Merge
  - UNION Transformations
    - Subselect Pruning
    - Distribution

Please submit your session feedback!

- Do it online at https://conferences.gse.org.uk/2021/feedback/2AQ

- This session is 2AQ

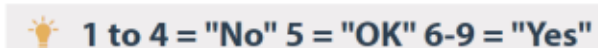
1. What is your conference registration number?



2. Was the length of this presention correct?

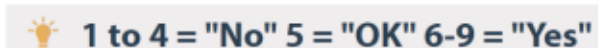
1 2 3 4 5 6 7 8 9

3. Did this presention meet your requirements?



1 2 3 4 5 6 7 8 9

4. Was the session content what you expected?



1 2 3 4 5 6 7 8 9

Become a member of GSE UK

- Company or individual membership available
- Benefits include:
  - GSE Annual Conference: Receive 5 free places + 2 free places for trainees
  - 20% discount on fees for IBM Technical Conferences
  - 20% on IBM Training Courses in Europe
  - 15% discount for IBM STG Technical Conferences in the USA
  - 20% discount on the fee for taking the Mainframe Technology Professional (MTP) exams
  - European events – via GSE HQ

- Contact membership@gse.org.uk for details