SQL Anywhere query optimizer

Glenn Paulley, Director, Engineering

8 March 2011

Sybase iAnywhere products

SQL Anywhere

- Full-function, small-footprint relational DBMS with support for triggers, stored procedures, materialized views, spatial data types, integrated full text search, intra-query parallelism, hot failover, ANSI SQL 2008 support including OLAP queries, multidatabase capability

Mobilink/SQL Remote

- Two-way data replication/synchronization technologies for replicating data through different mechanisms to support occasionally-connected devices

Ultralite, Ultralite J

- “fingerprint” database supports ad-hoc SQL on very small devices
- Ultralite J supports Blackberry

SYBASE iAnywhere

SQL Anywhere Query Processing

Query execution

- Utilizes data-flow operator trees (“Volcano”)
- Join methods: hybrid-hash, merge, nested-loop
  - Hash, nested-loop recursive joins
  - Merge, hash-based FULL OUTER JOIN
  - Merge, hash-based INTERSECT, EXCEPT
- Nested loop, hash-based semijoins, anti-semijoins
- Sort-, hash-based grouping, duplicate elimination
- Low-memory alternative plans
  - Identified by the optimizer at optimization time
  - Switch is performed at runtime if conditions warrant

Query execution operator trees

21.xml

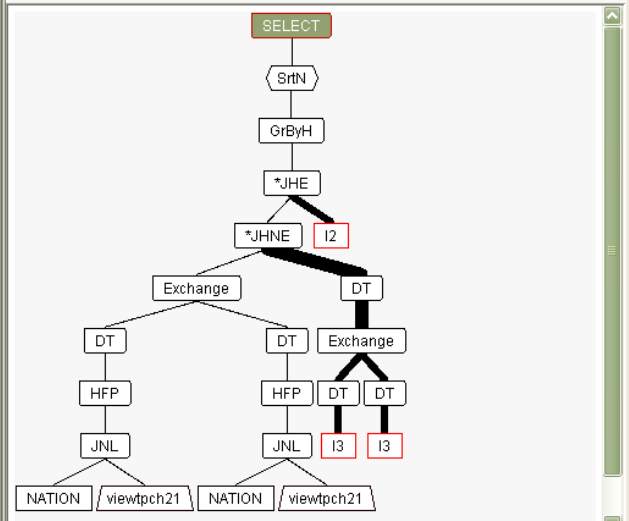
File Edit SQL Data Tools Window Help

SQL Statements

```
SELECT TOP 100 s_name, count(*) AS numwait --- vQ21
FROM supplier, lineitem l1, orders, nation
WHERE s_suppkey = l1.l_suppkey AND o_orderkey = l1.l_orderkey AND o_orderstatus = 'F' AND l1.l_receiptdate > l1.l_commitdate
    AND EXISTS (
            SELECT TOP 10 *
            FROM lineitem l2
            WHERE l2.l_orderkey = l1.l_orderkey AND l2.l_suppkey <> l1.l_suppkey )
    AND NOT EXISTS (
            SELECT DISTINCT l_orderkey
            FROM lineitem l3
            WHERE l3.l_orderkey = l1.l_orderkey AND l3.l_suppkey <> l1.l_suppkey AND l3.l_receiptdate > l3.l_commitdate )
    AND s_nationkey = n_nationkey
    AND n_name = 'CANADA'
GROUP BY s_name
ORDER BY numwait DESC, s_name
```

Results

Main Query



Details | Advanced Details

RowsReturned	100	Number of rows returned
PercentTotalCost	100	Run time as a percent of total query time
RunTime	1022.4	Time to compute the results
CPUTime	284.66	Time required by CPU
DiskReadTime	737.78	Time to perform reads from disk
DiskWriteTime	0	Time to perform writes to disk
DiskRead	1.9703e+006	Disk reads
DiskWrite	0	Disk writes

Optimizer statistics

	Value	Description
Costed subplans	142	Number of different enumeration strategies considered by the optimizer
Estimated cache pages	229383	Estimated cache pages available for this statement
CurrentCacheSize	1881820	Current cache size in kilobytes
isolation_level	0	Controls the locking isolation level
optimization_goal	All-rows	Optimize queries for first row or all rows
optimization_level	9	Controls amount of effort made by the query optimizer to find an access plan
optimization_workload	OLAP	Controls whether optimizing for OLAP or mixed queries
ProductVersion	10.0.0.3394	Product version
user_estimates	Override-magic	Controls whether to respect user estimates

Messages | Plan

Line 11 Column 54

SYBASE iAnywhere

Role of the optimizer

Select an efficient access plan (operator tree) for the given SQL statement

- typically queries, but UPDATE and DELETE statements also require optimization

Try to do so in an efficient amount of time

- requires tradeoffs between optimization and execution time
- finding the ‘optimal’ plan a misnomer; we typically only desire an adequate strategy, hence the goal is to avoid selecting poor strategies
- Simple SQL statements bypass the cost-based optimizer, and access plan generation uses a simple heuristic algorithm instead

Query optimizer

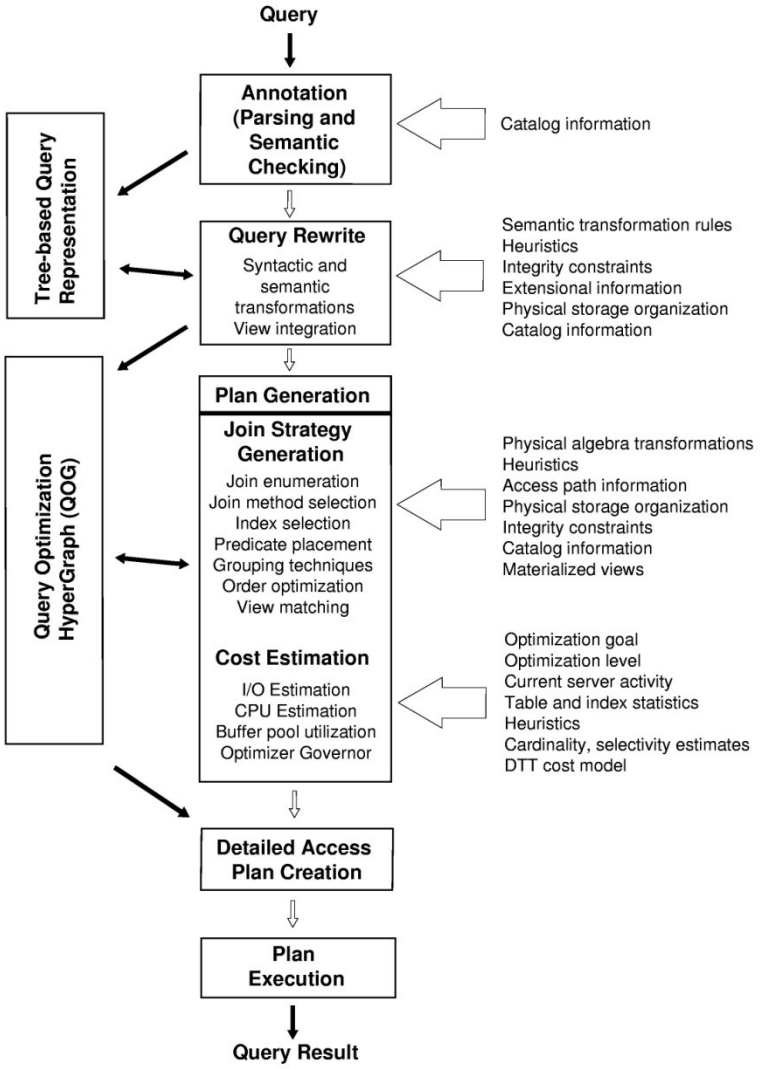
SQL Anywhere optimizes requests each time they are executed

- Optimizer takes into account server context:
  - Working set
  - Available cache
  - Number of available server threads
  - Amount of table/index pages already in cache
  - Values of procedure, host variables
- **Assumption: optimization is cheap**
  - SQL Anywhere uses a proprietary, branch-and-bound join enumeration algorithm that primarily constructs left-deep trees
  - Optimization process includes both heuristic and cost-based complex rewritings
- Advantages: plans are responsive to server environment, buffer pool contents/size, data skew; no need to administer ‘packages’

The optimization process in SQL Anywhere

- Parser converts query syntax into a parse tree representation
  - facilitates cost-based subquery rewritings
- Heuristic semantic (rewrite) optimization
- Parse tree converted into a query optimization graph
- Selectivity analysis
- Join enumeration, group-by, order optimization performed for each subquery
  - Includes index selection, parallelization, physical operator choices, cost-based rewrite optimizations, placement of Bloom filter constructors and predicates
- Post-processing of chosen plan
  - Conversion of optimization structures to execution plan
- Construct execution plan

Query optimization in SQL Anywhere



SYBASE iAnywhere

Optimization process overview

- **Three distinct phases**
  - Normalization phase
    - Normalizes query input into a canonical form
    - Performs heuristic-based query rewrite optimization
  - Preoptimization phase
    - Constructs necessary building blocks for all plan components
      - Termed a “query optimization graph”, a directed hypergraph
    - Selectivity estimation
    - Eliminates from consideration useless indexes, join methods, materialized views
  - Enumeration phase
    - Performs strategy enumeration using a branch-and-bound algorithm without memoization

Rewrite optimizations

- Uses the parse tree built during annotation phase
- Many are heuristic-based transformations
- Transformations are irreversible
- Examples of semantic transformations:
  - Unnecessary DISTINCT and row limit elimination
  - Rewriting subqueries as EXISTS predicates
  - Subquery unnesting
  - Predicate pushdown into GROUPed or UNION views and derived tables
  - Conversion of outer joins to inner joins
  - Outer and inner join elimination

Plan enumeration process overview

- Depth-first search over the space of left-deep strategies
- Algorithm includes order optimization, join method selection, cost-based index selection
  - There are n! join strategies for a fully connected join graph over n quantifiers (assuming only inner joins)
- The search space is an ordered forest of rooted ordered trees of partial and full access plans
- Space of possible access plans for a join query of degree n-1 is $O(n! J^{(n-1)}K^{(n)})$ where
  - J – number of physical join operators
  - K – number of access methods available for a quantifier (e.g., partial index scans, best plan of a derived table)
  - Complicated by
    - Support for left-outer and full-outer joins (possibly nested)
    - Preserved tables must precede null-supplying tables in any left- or right-outer join strategy
    - Index-only retrieval, materialized view matching

Join strategy enumeration

- Construct a hypergraph (nodes are quantifiers, edges are join predicates) to represent the statement
- Recall: building a left-deep tree
- Recursively:
  - Select a quantifier at position *n* by analyzing the join graph
    - “rank” candidate quantifiers according to their participation in any outer joins (preserved tables must precede null-supplying ones), cardinality, and so on
    - Existence of join graph edges between this quantifier and other quantifiers that precede it in the current join strategy prefix
    - Existence of edges between this quantifier and others yet to be placed in the strategy

Join strategy enumeration

- Then:
  - Select a join method in combination with a relevant index that matches join predicates and/or other highly selective predicates
  - Place any other local predicates that can now be evaluated into this partial plan
    - Incrementally cost this partial plan
  - **Key point**: cost is based on **a highly optimistic heuristic**: that each quantifier has up to ½ of the buffer pool at its disposal for the duration of the request

Join strategy enumeration

- Once a complete join strategy is enumerated, its cost can be determined exactly (with respect to buffer pool utilization)
  - Note: after a single strategy is enumerated, we have a complete, valid access plan (contrast this with dynamic programming)
- Other alternative plans are evaluated due to the recursive nature of the depth-first search
- Pruning takes place when a *prefix* of a join strategy is more costly than the “best” strategy costed so far
  - *Optimistic* nature of costing each join guarantees that useful strategies won’t be pruned

Information sources

- **Logical operator properties**
  - Includes cardinality estimation, size of a projection
  - Sources:
    - Simple column histograms
    - Index sampling
    - Temporary join histograms
    - Base table statistics
- **Physical operator properties**
  - Includes order properties, parallel execution properties, CPU costs, I/O costs
  - Sources:
    - Table and index statistics
    - Current workload, including potential degree of parallelization
    - Buffer pool statistics
    - Disk transfer time (DTT) model

Effect of heuristics

- The order of the generation of different access plans is critical
  - Finding the best plan early may significantly decrease overall optimization time
- The enumeration algorithm uses several heuristics based on the database schema and the characteristics of the query optimization graph to generate the next partial plan
- Property vectors are saved for each node in the current partial plan, hence backtracking uses these previously-computed property vectors to reduce the amount of computation

Optimizer governor

- In some cases the optimizer may have to choose between many possible plans that offer little, if any, cost improvement
- The optimizer's governor sets a maximum number of plan nodes that will be visited during enumeration
  - Based on the current setting of OPTIMIZATION_LEVEL
- Maximum number of nodes will be reduced automatically if the optimizer discovers a sufficiently cheap access plan
- Optimizer spends less effort on optimizing inexpensive queries

Example query

Example query (TPC-H Query 10):

- “Returned Item Reporting Query”
- List, in descending order of lost revenue, for a given calendar quarter, those customers who have returned any parts.
- Lost revenue is defined as
  - SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT))

Example

Table	Rows	Pages	Rows/page
ORDER	300,000	7,928	37.84
LINEITEM	1.1M	30,072	39.9
NATION	25	1	25
CUSTOMER	30,000	816	36.76

```
SELECT C_CUSTKEY, C_NAME,
            SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
        C_ACCTBAL, N_NAME, C_ADDRESS, C_PHONE, C_COMMENT
    FROM   CUSTOMER, ORDER, LINEITEM, NATION
    WHERE  C_CUSTKEY = O_CUSTKEY
            AND  L_ORDERKEY = O_ORDERKEY
            AND  O_ORDERDATE >= DATE( '1995-01-01')
            AND  O_ORDERDATE < DATEADD(MONTH, 3, '1995-01')
            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
```

SYBASE iAnywhere

Enumeration of the join strategy

- Table ORDER has the highest rank, due to its join graph outdegree
  - Consider sequential scan first, since predicates on O_ORDERDATE are not selective enough
- (Recursively) place LINEITEM as the next table, as it has the highest remaining rank
  - Indegree is 1 (Join predicate with ORDER)
  - Outdegree is 0, but there is a filtering predicate on L_RETURNFLAG (selectivity: 24.67%)
    - Choose hash join as the physical join method
- (Recursively) place the remaining quantifiers
- Compute the cost of the complete plan

Enumerating subsequent strategies

- Other permutations of the join strategy are considered in a depth-first manner
- Optimizer governor controls what proportion of the space is searched for each particular quantifier position
- Partial strategies are pruned if their costs exceed the cost of the cheapest plan discovered thus far
  - Uses the ½ available buffer pool heuristic for cost estimation
  - Optimizer takes server context into account when determining physical access paths, join methods; these include:
    - amount of query memory available
    - number of currently active requests
    - amount of table data already in the buffer pool
    - server multiprogramming level

Access plan generation

Access plans can vary depending on:

- OPTIMIZATION_GOAL setting (FIRST-ROW or ALL-ROWS)
- Cursor type
- Amount of buffer pool available at optimization time
- Percentage of table data already in the database cache (buffer pool)
- OPTIMIZATION_LEVEL setting (1-15, controls governor)
- OPTIMIZATION_WORKLOAD setting
- Isolation level of cursor or table
- Capabilities of the underlying hardware (DTT cost model)

Summary

- SQL Anywhere's branch-and-bound join enumeration algorithm is very efficient, even for large numbers of joins
- Uses very little RAM – most of the necessary data structures are allocated on the stack
- Ideal for embedded systems or for handheld devices running Windows Mobile
  - Optimization takes into account server context
  - Server self-configures automatically in response to changing workload, other demands on the OS
    - Optimizer needs to adjust enumeration parameters accordingly
  - We have tested our server with queries having a join degree of 500 on a Windows Mobile device with a 1GB database instance


SYBASE®
iAnywhere®