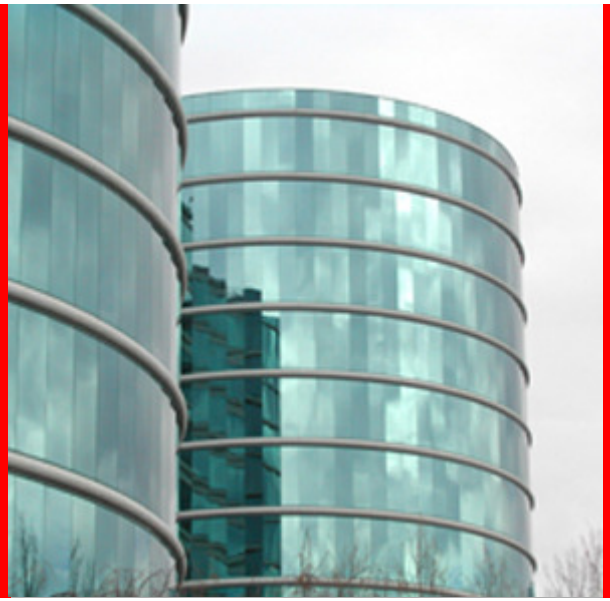


ORACLE®

# Supporting Time-Constrained SQL Queries in Oracle

Ying Hu, Seema Sundara, Jagannathan Srinivasan
Oracle New England Development Center
One Oracle Drive, Nashua, NH 03062

# Talk Outline

- The Problem and Our Approach
- Time-constrained SQL Queries
- Supporting Time-constrained SQL Queries
- Performance Study
- Conclusions

# The Problem and Our Approach

# The Problem

- Databases are growing
  - Giga Bytes → Tera Bytes → Peta Bytes
- Arbitrarily complex queries
  - Using SQL (JOINs, GROUP BY, ORDER BY, etc.)

Resulting in

➔Long running SQL Queries

➔Unpredictable Query Response Time

# The Problem


TIME IS MONEY

- Thus, the current scheme of issuing a SQL query and letting it take whatever time (and resources) to complete is *unsatisfactory* especially when the user is *constrained by time*.

# Prior Approaches for Time Constraints

- **Return first few (or top-k) rows**

  [SIGMOD 1997] M. Carey, D. Kossmann:
  On saying "enough already!" in SQL.

  - Augment the query with a range predicate

    [VLDB 1999] S. Chaudhuri, L. Gravano:
    Evaluating Top-k Selection Queries.

    [VLDB 1999] D. Donjerkovic, R. Ramakrishnan:
    Probabilistic Optimization of Top N Queries.

  - For joins, generate results ordered on a rank function

    [VLDB J. 2004] I. F. Ilyas, W. G. Aref, A. K. Elmagarmid:
    Supporting Top-k Join Queries in Relational Databases.

  - In Oracle,
    - ROWNUM clause to express top-k queries
    - The hint /*+ FIRST_ROWS */ to indicate that query be optimized for first few rows

# Prior Approaches for Time Constraints

- **Compute Approximate Results**
  - return approximate results by use of sampling, histograms etc.
  - employed for online aggregation, includes estimating errors in reported results (e.g. confidence intervals)

    [SIGMOD 1997] J. M. Hellerstein, P. J. Haas, H. J. Wang: Online Aggregation.

    [DMKD 2000] J. M. Hellerstein, R. Avnur, V. Raman: Informix under CONTROL: Online Query Processing.

  - In Oracle, SAMPLE clause to indicate only portion of a table be used

# The Problem Remains

- The onus is on user to employ these approaches intelligently!

Not easy to translate a time constraint to equivalent

- a first-few (top-k) rows query

or

- an approximate query

# Our Approach

- Introduce a time-constraint clause to SQL SELECT Query that specifies
  - Type of constraint: Soft or Hard
  - Time limit: in seconds
  - Acceptable Nature of results: partial or approximate

- Let the Database System do the needed transformation to execute the query in specified time limit

# Our Approach

- The transformed query returns either
  - first-few (top-k) rows, or
  - approximate results
- Both of which are expected (guaranteed) to complete in the specified time limit for soft (hard) time constraint

# Time-constrained SQL Queries

# A New Time Constraint Clause

SELECT … FROM …

WHERE …

GROUP BY … HAVING …

ORDER BY …

**[ [ SOFT |HARD] TIME CONSTRAINT (T)**
**[WITH { APPROXIMATE | PARTIAL} RESULT ]**

**];**

# An Example

- A time constrained SQL query

```
SELECT AVG(salary)
FROM employees
SOFT TIME CONSTRAINT (50)
  WITH APPROXIMATE RESULT;
```

- Query after rewrite may be transformed into

```
SELECT AVG(salary)
FROM employees SAMPLE BLOCK (10);
```

# Soft Time Constraint Definition

**Definition**:

*A query Q with a soft time constraint of t sec*

$\Rightarrow T_{estimated\_by\_optimizer}(Q') \text{ BETWEEN } t\text{-}d \text{ AND } t$,

*where d is a small time unit and*
*Q' is the transformed query*

# Hard Time Constraint Definition

**Definition**:

*A query Q with a hard time constraint of t sec*

$\Rightarrow T_{elapsed}(Q') <= t,$ ,

*where Q' is the transformed query*

# Functions for Estimating Aggregates and Corresponding Confidence Interval Values

- For queries returning approximate results
  - Provide functions for estimating aggregates over the entire table
    - *estimatedSum, estimatedCount, estimatedAvg*
  - Provide ancillary functions to return the confidence interval associated with each aggregate function
    - *sumConfidence, countConfidence, avgConfidence*

    The confidence interval functions are based on Central Limit Theorem or Hoeffding's inequality

    [SSDBM 1997] P. J. Haas: Large-Sample and Deterministic Confidence Intervals for Online Aggregation

# Functions for Estimating Aggregates and Corresponding Confidence Interval Values

Example

```
SELECT   COUNT(*) SAMPLECOUNT,
         estimatedCount(*) ESTIMATEDCOUNT,
         countConfidence(*, 95) COUNTCONFIDENCE
FROM employees  SOFT TIME CONSTRAINT(5)
                WITH APPROXIMATE RESULTS;
```

Result

| SAMPLECOUNT | ESTIMATEDCOUNT | COUNTCONFIDENCE |
|---|---|---|
| 207000 | 1200900 | 14000 |

# Supporting Time Constrained SQL Queries

# Scheme for Supporting Soft-Time Constraint Queries

- Basic Idea:
  - Transform the input query by augmenting either with
    - ROWNUM clause that *reduces the result set size*, OR
    - SAMPLE clause that *reduces the data blocks scanned OR the intermediate result size returned* from the referenced table(s)
  - The resulting query is executed, which is expected to finish sooner
  - The challenge:
    - If ROWNUM clause used - *estimating result set cardinality*
    - If SAMPLE clause used - *estimating* table sample size, as well as *deciding the list of tables for which sampling should be done* (in case of multi-table queries)
    - Ensuring that the estimated time for resulting query satisfies the time-constraint

# Query Transformation: Sampling Referenced Tables

IF original query is

```
SELECT … FROM T
WHERE …
```

THEN the transformed query becomes

```
SELECT … FROM T SAMPLE BLOCK(n)
 WHERE …
```

# Estimating Sample Size

- The function $f_Q$, which represents time to execute query Q depends on sample size s.
- Thus, $f_Q(s) = t$, where t is the specified time-constraint.
- The desired s is a root of equation

  $f_Q(s) - t = 0$

  and is obtained using a root finding algorithm

- Note: Oracle's cost-based optimizer' EXPLAIN PLAN facility is used to estimate $f_Q(s)$ for a given s.

# Estimating Sample Size: Details

1. Obtain estimated query time (by consulting optimizer) say $T_Q$ for original query Q
2. If $T_Q < t$ then STOP. No transformation needed
3. If $T_Q > t$ then obtain estimated query time $T_{Q'}$, where Q' is augmented query with minimum sample size
4. If $T_{Q'} > t$ then return 'ERROR: NEED MORE TIME'.
5. Iterate (using root-finding algorithm)

   till the estimated time is BETWEEN t-d AND t
5. Return the current s

# Sampling Based Query Transformation for Multi-table Joins w/ Foreign Keys

- For table joined via foreign key add sampling clause only to the largest fact table (Aqua System from Bell Labs)
- It is because a uniform random sample over foreign-key joins of tables can be achieved through a uniform random sampling over the largest fact table and then joining with other dimension tables

  [SIGMOD 1999] S. Acharya, et al :

  Join Synopses for Approximate Query Answering.

# Sampling Based Query Transformation for Multi-table Joins w/o Foreign Keys

- The goal is to have as many resulting rows as possible, or have as many rows as possible in the resultant joins for aggregate queries
- Thus, maximize $f_1 * f_2$, where $f_1$ and $f_2$ are the sample sizes for the two tables
- Case 1: Nested Loop Join:

  It can be proved that the sample clause should be put into only the outermost relation, i.e. $f_2 = 1$, no sampling over the inner relations

# Sampling Based Query Transformation for Multi-table Joins w/o Foreign Keys

- Case 2: Hash Join:
  - Compute sampling size $f_1$ and $f_2$ such that

    $f_1 * T_1 = f_2 * T_2$ ,

    where $T_1$ and $T_2$ are times used to process the two tables being joined because this will maximize $f_1 * f_2$
- Case 3: Sort-Merge Join:
  - Since sort has a time complexity of O(nlogn), there is no easy solution for sort-merge join. We adapt the above technique of making $f_1 * T_1 = f_2 * T_2$

# Sub-query Processing

- SELECT *
  FROM employees outer
  WHERE outer.salary >
  (SELECT AVG(inner.salary)
  FROM tax_return inner)
  SOFT TIME CONSTRAINT (10);
- Try not to push sample clause into the sub-query, because it can cause an approximate predicate
- Otherwise, the time allocated to each stage is determined through linear interpolation

# Scheme for Supporting Hard-Time Constraint Queries

- Basic Idea:
  - Transform the input query by treating the specified time limit as soft-constraint
  - The estimated time for the transformed query meets the specified time limit
  - Generate execution plan and use the estimated time information for various operations to associate timers as follows:
    - A timer for top-level operation with time set to specified time limit
    - A timer for every blocking sub-operation with time set to estimated time for corresponding operation in execution plan

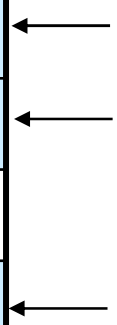
# Scheme for Supporting Hard-Time Constraint Queries: Example

- TPC-H Q14: Promotion effect query

```
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 lineitem, part
WHERE l_partkey = p_partkey AND
  l_shipdate >= date '1995-09-01' AND
  l_shipdate < date '1995-09-01' + interval '1' month
HARD TIME CONSTRAINT (300);
```

# Scheme for Supporting Hard-Time Constraint Queries: Example

| Id | Operation | Name | … | Estimated Time |
|---|---|---|---|---|
| 0 | SELECT STATEMENT | | | 300 |
| 1 | SORT AGGREGATE | | | 300 |
| 2 | HASH JOIN | | | 300 |
| 3 | TABLE ACCESS FULL | LINEITEM | | 270 |
| 4 | TABLE ACCESS FULL | PART | | 10 |



- Timers
  - Id 0: Top level time set to 300 seconds
  - Id 1: Blocking sub-operation timer set to 300 seconds
  - Id 3: Blocking sub-operation timer set to 270 seconds

# Leveraging Oracle's Cost-based Optimizer

- Object Statistics:
  - Number of blocks, number of rows for tables
  - Height of a B-tree indexes, etc.
- System Statistics:
  - Average number of CPU cycles/sec
  - Average time to read a single block (random read)
  - Average time to read multi-blocks (sequential read), etc.

- EXPLAIN PLAN
  - Utilizes Statistics collected to calculate CPU and I/O costs for each access method in a SQL query
  - It returns optimal execution plan as well as estimated time for query execution

# Performance Study

# Experiments: TPC-H

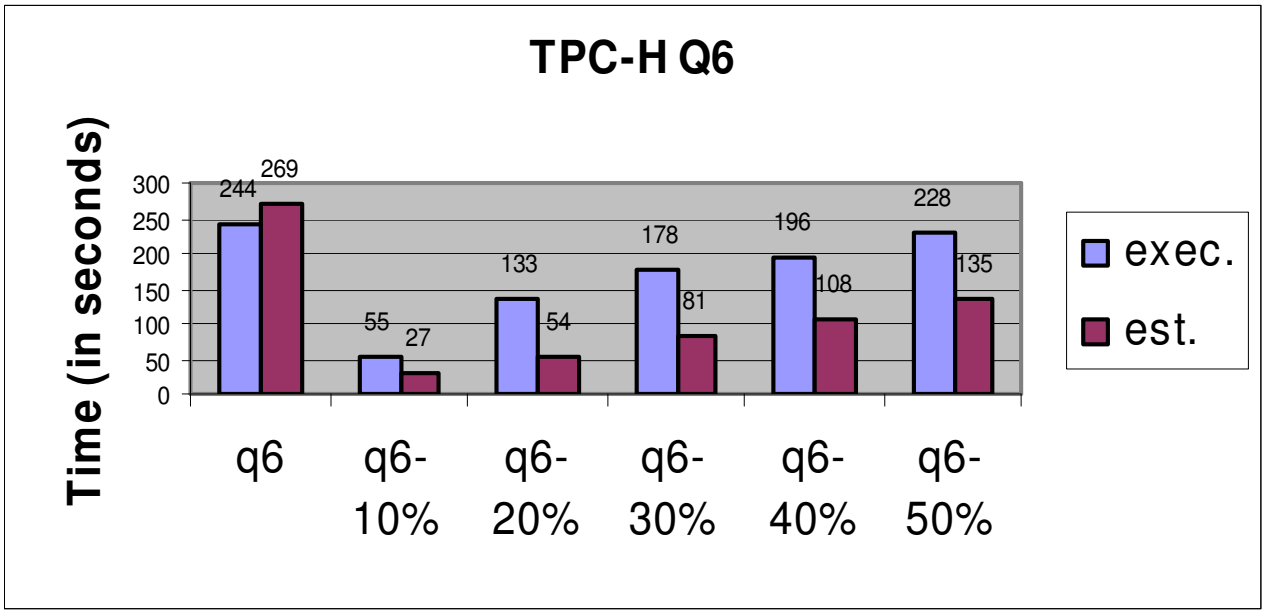
- Platform:
  - Intel P4 3.0Ghz with Hyper-Threading, 2GB main memory, and 80GB hard disk
  - Redhat Enterprise Linux 3 and Oracle Database 10g Release 2 Enterprise Edition
- Key Database Parameters:
  - db_block_size=8192, db_cache_size=160M
- Data Set
  - TPC-H database size of about 10GB, consisting of 8 tables
  - LINEITEM table is the biggest and has ~60 million rows
  - ORDERS table is the second largest with 15 million rows

# Single Table Query with Aggregates

- TPC-H Q6: This query considers all the line items shipped in a given year with discounts between a ± 0.01 of DISCOUNT=0.06

```
SELECT SUM(l_extendedprice * l_discount) AS revenue
FROM lineitem
WHERE l_shipdate >= date '1994-01-01'
  AND l_shipdate < date '1994-01-01' + interval '1' year
  AND l_discount between 0.06 - 0.01 and 0.06 + 0.01
  AND l_quantity < 24;
```

# Single Table Query with Aggregates




- Time constraints chosen: 10%, 20%, ... of original query estimated time
- Transformed query uses sampling
- Elapsed time > Estimated Time (due to less than expected use of multi-block I/O)
- Time does decrease as the user specifies smaller time-constraints

# Sum, Estimated Sum, & Confidence Interval

| % of time | SUM | estimatedSum | sumConfidence (confidence interval) |
|---|---|---|---|
| 10% | 113894821 | 1228484983 | 49916216 |
| 20% | 243045023 | 1230244879 | 34194884 |
| 30% | 370097887 | 1228624043 | 27692357 |
| 40% | 489547623 | 1228986671 | 24081572 |
| 50% | 617119157 | 1229137335 | 21449857 |
| 100% | 1230113636 | N/A | N/A |

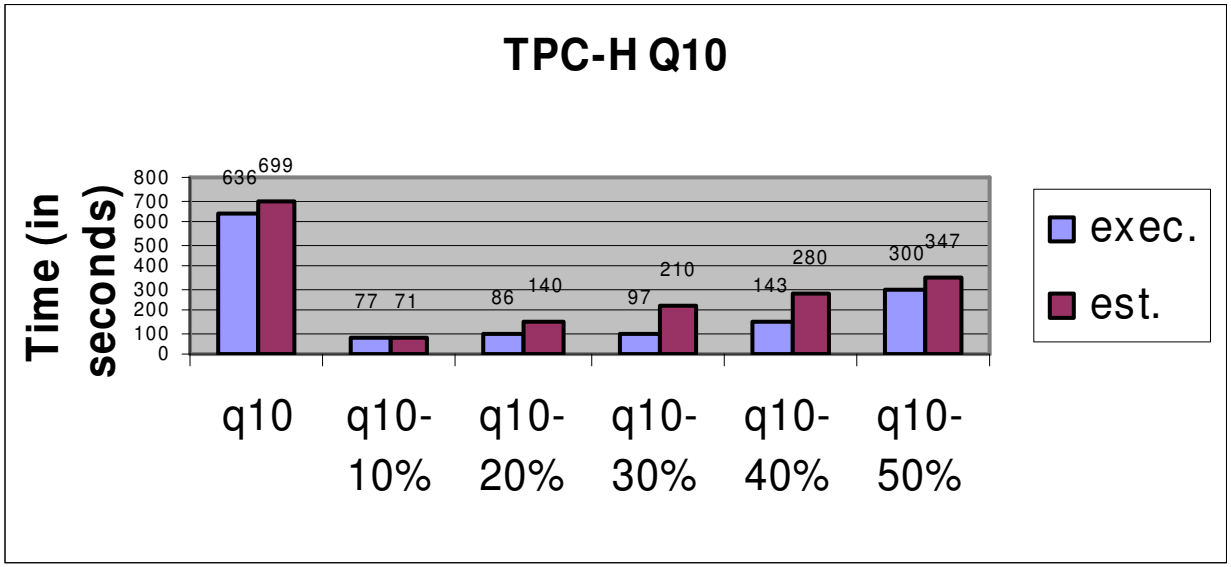
95% Confidence Interval computing using Hoeffding-based bounds

# Four Table Join Query with GROUP BY and ORDER BY

- TPC-H Q10: Returned Item Reporting Query

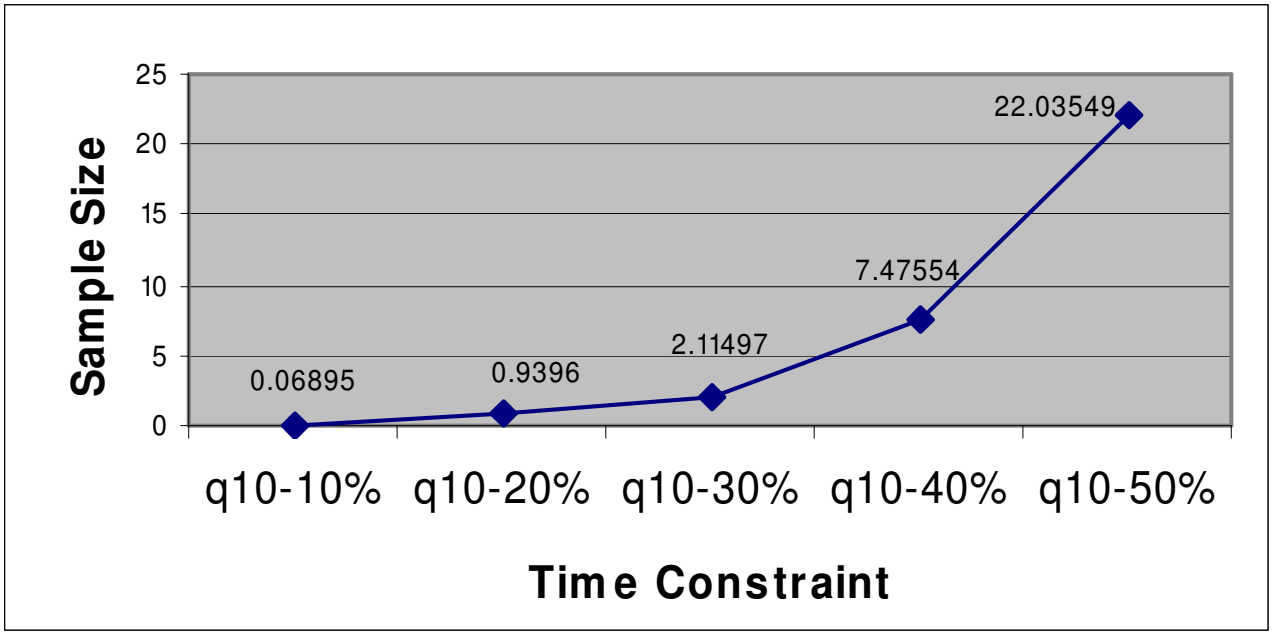
```
SELECT c_custkey,
    c_name,
    sum(l_extendedprice * (1 - l_discount)) AS revenue,
    c_acctbal, n_name, c_address, c_phone, c_comment
FROM customer, orders, lineitem, nation
WHERE c_custkey = o_custkey AND
          l_orderkey = o_orderkey AND
          o_orderdate >= date '1993-10-1' AND
          o_orderdate < date '1993-10-1' + interval '3' month 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;
```

# Four Table Join Query with GROUP BY and ORDER BY




- Only Lineitem table sampled as it is the largest table and has a foreign key reference to O_ORDERKEY
- Time constraint clause is effective in reducing the execution time

# Four Table Join Query with GROUP BY and ORDER BY




- The estimated sample size to meet the time constraint for various time-constraint queries
- In this case the maximum number of iterations required to estimate the sample size is 10. However, the total overhead for estimating sample size is quite small (< 0.5 sec)

# Conclusions and Future Work

- Time-constrained SQL queries must be supported in database systems. It can leverage work in the following areas:
  - top-k query optimization, approximate query processing, and error estimation
  - plus, the capabilities of cost-based optimizer, namely, the optimal plan generation, and accurate estimation of the query execution time
- Both support for soft and hard time-constraint were considered
- The experimental study conducted (on a prototype implementation using Oracle) with the TPC-H dataset demonstrates the effectiveness of time-constrained SQL queries

- In future, we plan to explore
  - tighter integration of the proposed techniques
  - the feasibility and effectiveness of supporting hard time constraints


QUESTIONS
ANSWERS