An Overview of Query Optimization

Chapter 11
Section 11.1, 11.2 and 11.3 (till page 415)

Query Evaluation

• **Problem:** How to execute a given SQL query?
  – SQL is declarative, a SQL query does not specify how to execute the query
  – A relational algebra expression is procedural
  – contains operations that can be executed in a specific order to execute the query

• **Solution:** Convert SQL query to an equivalent relational algebra
  – But which equivalent expression is best?

Naive Conversion

SELECT DISTINCT TargetList
FROM R1, R2, …, RN
WHERE Condition
  is equivalent to
  \(\pi_{\text{TargetList}}(\sigma_{\text{Condition}}(R1 \times R2 \times \ldots \times RN))\)
  but this may imply a very inefficient query execution plan.
  
Example:
  \(\pi_{\text{Name}}(\sigma_{\text{Id=ProfId } \land \text{CrsCode}='CS532'}(\text{Professor} \times \text{Teaching}))\)
  • Result can be < 100 bytes
  • But if each Teaching has 5K tuples of size 10B and Professor has 1K tuples of size 50B, then we end up computing an intermediate result Professor \times Teaching of size 300M before shrinking it down to just a few bytes

**Problem:** Find an equivalent relational algebra expression that can be evaluated “efficiently”
Query Processing Architecture

SQL Query → DNL Parser → Relational Algebra Expression → Query Optimizer → Query Execution Plan → Query Result

Query Optimizer

- Uses heuristic algorithms to evaluate relational algebra expressions. This involves:
  - estimating the cost of a relational algebra expression
  - transforming one relational algebra expression to an equivalent one
  - choosing access paths for evaluating the subexpressions
- Output: Query Execution Plan
  - Tree of operators, with choice of algorithm for each op
  - Operators: SELECT, PROJECT, JOIN…
- Query optimizers do not “optimize” – just try to find “reasonably good” query plans

Topics

- Introduction
- Optimization Using Relational Algebra Equivalence
- A Quick Look at Implementation of Relational Operators
- Optimization Using Cost Estimates
- Query Plans in Oracle

Equivalence Preserving Transformations

- To transform a relational expression into another equivalent expression we need transformation rules that preserve equivalence
- Each transformation rule
  - Is provably correct (i.e., does preserve equivalence)
  - Has a heuristic associated with it
Selection and Projection Rules

• Break complex selection into simpler ones:
  \[ \sigma_{Cond1 \land Cond2}(R) \equiv \sigma_{Cond1}(\sigma_{Cond2}(R)) \]

• Break projection into stages:
  \[ \pi_{attr}(R) \equiv \pi_{attr}(\pi_{attr'}(R)), \quad \text{if } attr \subseteq attr' \]

• Commute projection and selection:
  \[ \pi_{attr}(\sigma_{Cond}(R)) \equiv \sigma_{Cond}(\pi_{attr}(R)), \quad \text{if } attr \supseteq \text{all attributes in } Cond \]

Pushing Selections and Projections

• \[ \sigma_{Cond}(R \times S) \equiv R \downarrow_{\sigma_{Cond}} S \]
  - \( Cond \) relates attributes of both \( R \) and \( S \)
  - Reduces size of intermediate relation since rows can be discarded sooner

• \[ \sigma_{Cond}(R \times S) \equiv \sigma_{Cond}(R) \times S \]
  - \( Cond \) involves only the attributes of \( R \)
  - Reduces size of intermediate relation since rows of \( R \) are discarded sooner

• \[ \pi_{attr}(R \times S) \equiv \pi_{attr}(\pi_{attr'}(R) \times S), \quad \text{if } attributes(R) \supseteq attr' \supseteq attr \cap attributes(R) \]
  - Reduces the size of an operand of product

Commutativity and Associativity of Join
(and Cartesian Product as Special Case)

• Join commutativity: \( R \bowtie S \equiv S \bowtie R \)
  - used to reduce cost of nested loop evaluation strategies (smaller relation should be in outer loop)

• Join associativity: \( R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T \)
  - used to reduce the size of intermediate relations in computation of multi-relational join - first compute the join that yields smaller intermediate result

• N-way join has \( T(N) \times N! \) different evaluation plans
  - \( T(N) \) is the number of different binary trees with \( N \) leaf nodes
  - \( N! \) is the number of permutations

• Query optimizer cannot look at all plans (might take longer to find an optimal plan than to compute query brute-force). Hence it does not necessarily produce optimal plan
  - E.g., for \( N = 4, T(4) = 5, N! = 24 \), number of plans = 120
  - For \( N = 5, T(5) = 14, N! = 120 \), number of plans = 1680

Equivalence Example

• \[ \sigma_{C1 \land C2 \land C3}(R \times S) \equiv \sigma_{C1}(\sigma_{C2}(\sigma_{C3}(R \times S))) \]
  \[ \equiv \sigma_{C1}(\sigma_{C2}(R) \times \sigma_{C3}(S)) \]
  \[ \equiv \sigma_{C2}(R) \bowtie_{\sigma_{C1}} \sigma_{C3}(S) \]

assuming \( C2 \) involves only attributes of \( R \), \( C3 \) involves only attributes of \( S \), and \( C1 \) relates attributes of \( R \) and \( S \)
Query Tree

- A relational algebra expression can be viewed as a tree
- Binary operators (e.g., join) have two children
- Unary operators (e.g., select, project) have one child
- Leaf nodes correspond to base relations
- For a given SQL query, different relational algebra expressions will correspond to different query trees
- Query trees are used to evaluate the cost of executing the query using the corresponding relational algebra expression

Example 1

```
SELECT P.Name
FROM Professor P, Teaching T
WHERE P.Id = T.ProfId
AND P.DeptId = 'CS'
AND T.Semester = 'F1994'
```

\[ \pi_{\text{Name}}(\sigma_{\text{DeptId}=\text{CS}} \land \text{Semester}=\text{F1994})(\text{Professor} \bowtie \text{Teaching}) \]

Example 2

\[ \pi_{\text{Name}}(\sigma_{\text{DeptId}=\text{CS}})(\text{Professor}) \bowtie \sigma_{\text{Id}=\text{ProfId}}(\pi_{\text{Semester}=\text{F1994}}(\text{Teaching})) \]

Example 3

\[ \pi_{\text{Name}}(\sigma_{\text{Semester}=\text{F1994}})(\sigma_{\text{DeptId}=\text{CS}})(\text{Professor}) \bowtie \sigma_{\text{Id}=\text{ProfId}}(\text{Teaching}) \]
Example 4

$$\pi_{\text{Name}} \sigma_{\text{Dept} = 'CS'} ((\text{Professor}) \bowtie \sigma_{\text{Semester} = 'F1994'} (\text{Teaching}))$$

Heuristics for Using Algebraic Equivalences

1. Break up conjunctions in selection conditions
2. Propagate selections as far inside the query as possible
3. Combine Cartesian product operations with selections to form joins
4. Rearrange the order of join operations
   - Justification will come later
5. Propagate projections as far inside the query as possible
6. Identify the operations that can be processed in the same pass
   - Justification will come later

Query Execution Plan

- *Query execution plan* is a query tree but with concrete evaluation methods attached to each operation in the query tree
  - E.g., Query plan for query tree in Example 1. Join performed using Block-nested loops algorithm

Evaluating Alternative Query Plans

- We have seen that for a given query, there can be multiple relational expressions and hence multiple query trees
- Typically for a given query tree there are multiple possible Query Plans
  - E.g., choice of join algorithms
- To choose a “good” Query Plan, the DBMS’s Query Optimizer estimates the cost of Query Plans
- We will take a quick look at methods to implement relational operators and use this knowledge in examples of cost estimation
Topics

• Introduction
• Optimization Using Relational Algebra Equivalence
• A Quick Look at Implementation of Relational Operators
  – Cost metric
  – Cost computation for example operators
• Optimization Using Cost Estimates
• Query Plans in Oracle

Evaluating Relational Algebra Operators

• Note: An in-depth study of this topic requires knowledge of Chapter 9 and Chapter 10
• I expect that students have some knowledge of Chapter 9 (from CSCI 2150). However, I don’t expect you to read Chapter 10
• Whatever is needed from Chapter 10 will be covered in the slides

Physical Data Organization

• Database data is generally stored on disks rather than in main memory
  – Reasons: size, cost, volatility
• For CPU to process data, I/O operations must be executed to bring the data from the disk to main memory
• Disk block: unit of transfer of data for each I/O operation
  – Size of disk block is a parameter set when database is first created
  – E.g., size of data block in the database server being used in the class = 8KB

Data in Main Memory

• I/O time, i.e., the time to access data from disk, is typically orders of magnitude larger than the time CPU takes to process the data
• To keep the DBMS performing well, a large area of memory is allocated as a buffer to cache data accessed from disk
  – Size would be in gigabytes for large databases
  – If the desired data is in the cache, then I/O operation doesn’t need to be performed
• Page: term generally used for unit in memory that corresponds to a data block
  – Page size (in memory) = Disk block size (on disk)
Cost of a Query Plan

- Executing a query plan involves:
  - Reading (relevant) data blocks of the tables from disk to memory as needed by the specific operations specified in the query plan
  - Unless data is already in the cache
  - Processing of the data by the CPU
- Since I/O costs are much higher than CPU costs, the cost estimate generally ignores CPU costs
- Therefore
  - Estimated Cost of Query Plan = Cost of I/O

Cost of Query Plan (cont’d)

- Estimating I/O cost of operations in the Query Plan requires statistics or estimates about the tables in the query tree
  - How big are these tables?
  - What indexes are defined on these tables?
  - How much of the table/index will need to read?
- Such data is termed *meta-data*
  - Since it is data about data

Example: Metadata on Tables

- Professor (Id, Name, DeptId)
  - Size: 200 pages, 1000 rows, 50 departments
  - Indexes:
    - Clustered, 2-level B+ tree on DeptId
    - Hash on Id
- Teaching (ProfId, CrsCode, Semester)
  - Size: 1000 pages, 10,000 rows, 4 semesters
  - Indexes:
    - Clustered, 2-level B+ tree on Semester
    - Hash on ProfId

Statistics & Catalogs

- **System Catalog** contains meta data about tables and indexes
- The statistics may include:
  - # tuples (NTuples) and # pages (NPages) for each table
  - # distinct key values (NKeys) and NPages for each index
  - Index height, low/high key values (Low/High) for each tree index
- Statistics are updated periodically
  - Updating statistics whenever data changes is too expensive
  - Lots of approximation anyway, so slight inconsistency ok
Example DBMS: Oracle

- Data about Tables kept in:
  - DBA_TABLES
  - USER_TABLES: for the User logged on
  - DBA_TAB_COLUMNS (or USER_TAB_COLUMNS)
- Data about Indexes kept in:
  - DBA_INDEXES
  - USER_INDEXES: for the User logged on
- Structural as well as statistical data is stored
  - Structural data: e.g., number and data type of columns
  - Statistical data: e.g., size of table in data blocks
- Statistical data may require explicit collection

Topics

- Introduction
- Optimization Using Relational Algebra Equivalence
- A Quick Look at Implementation of Relational Operators
  - Cost metric
  - Cost computation for example operators
- Optimization Using Cost Estimates
- Query Plans in Oracle

Selection: Access Path

- Executing the selection operator requires retrieving the tuples that match the query
- An access path is a method of retrieving tuples
- One access path available for any table is file scan
  - Read every page in the file for the table into memory and discard the tuples that don’t match the query
  - If a table has $P$ pages, cost of a (full) file scan is $P$

Matching Index

- If a matching index is available, then that index can be the access path for a table in the query
- The combination of attributes on which an index is defined are termed the search key of the index
  - An overloading of the term key
  - Search key may not be a candidate key
- A tree index matches (a conjunction of) terms that involve only attributes in a prefix of the search key
  - E.g., Tree index on $<a, b, c>$
    - matches the selection
      - $a=5 \land b=3$
        - Does not match
          - $b=3$
Matching Index (cont’d)

- Terms that match an index reduce the number of tuples retrieved.
- Other terms are used to discard some retrieved tuples, but do not affect the number of tuples/pages fetched.
- Consider a selection with two terms
  - \( \text{DeptId}=5 \text{ AND Name}='\text{Joe}' \)
  - A B+ tree index on \( \text{DeptId} \) can be used
    - then, \( \text{Name}='\text{Joe}' \) must be checked for each retrieved tuple
- Cost of selection using a tree index depends on:
  - Height of the index
  - Whether the index is clustered or not
  - Number of tuples matching the terms in selection

One Approach

- Find the most selective access path, retrieve tuples using it, and apply any remaining terms that don’t match the index:
  - Most selective access path: An index or file scan that we estimate will require the fewest page I/Os
  - Terms that match this index reduce the number of tuples retrieved.

Example: Selection Using An Index

```
SELECT *
FROM  Professor P
WHERE  P.Name < 'C%'
```

- Professor (\( \text{Id}, \text{Name}, \text{DeptId} \))
  - Size: 200 pages, 1000 rows, 50 departments
  - Indexes: Clustered B+ tree on Name
- Example:
  - Assume uniform distribution of names
  - About 10% of tuples qualify, i.e., 100 rows, 20 pages
  - With clustered index, cost is little more than 20 I/Os
  - “little more” = cost of finding the qualifying leaf node
  - If unclustered index, cost could be much higher

Join

- Conceptually join can be thought of as one FOR loop over one table embedded within a FOR loop over another table
- Example: An equijoin of tables \( R \) and \( S \) over attributes \( R.x \) and \( S.y \) requires
  ```
  FOR each tuple \( r \) in \( R 
  FOR each tuple \( s \) in \( S 
  IF \( r.x == s.y \)
  add \( r, s \) to result
  ```
- Typically DBMS’s implement more than one join algorithms
  - Nested loop join (and its variations, e.g., Block and Index NL)
  - Sort-merge join
  - Hash join
Example: Block Nested Loop Join

- Fill buffer with blocks of table $R$, but leave two pages unused
- Now scan through $S$ by bringing a block into memory and joining it with blocks of $R$ in memory
- Write the result in the other unused page
- Cost of processing the input data:
  - Let $R$ be the number of pages in table $R$, $S$ be the number of pages in table $S$, $B$ be the size of the buffer
  - If $B - 2 \geq R$, then Cost = $R + S$
  - If $B - 2 < R$, then Cost = $R + S\cdot\text{ceil}(R/B - 2)$
- Actually the second formula also holds for the first case
- Observation: Cost is lower if the smaller table is the outer table

Example: Index-Nested Loop Join

- Instead of scanning $S$, use an index on $S$ with search key $y$, to find rows of $S$ that match rows in $R$
  - FOR each tuple $r$ in $R$
    - use index to find all tuples $s$ in $S$ satisfying $r.x = s.y$
    - add such $<r, s>$ to result
- Cost of processing the input data:
  - Important factors in cost estimate include:
    - How many tuples in $S$ match each tuple in $R$, and
    - The type of index
    - B+ tree or Hash
    - Clustered or Unclustered

Cost: Index-Nested Loop Join

- Consider we have a clustered B+ tree index
  - $R$ = number of pages in table $R$
  - $S$ = number of pages in table $S$
  - $t_R$ = number of tuples in $R$
  - $l$ = number of I/Os needed to retrieve the matching leaf node
  - $p_s$ = number of pages in $S$ with matching tuples for one tuple in $R$
    - $p_s$ is likely to be small
  - Cost = $R + t_R \cdot (l + p_s)$

Topics

- Introduction
- Optimization Using Relational Algebra Equivalence
- A Quick Look at Implementation of Relational Operators
- Optimization Using Cost Estimates
- Query Plans in Oracle
Example: Metadata on Tables

• **Professor** (Id, Name, DeptId)
  – Size: 200 pages, 1000 rows, 50 departments
  – Indexes:
    • Clustered, 2-level B+tree on DeptId
• **Teaching** (ProfId, CrsCode, Semester)
  – Size: 1000 pages, 10,000 rows, 4 semesters
  – Indexes:
    • Clustered, 2-level B+tree on Semester

Example 1

\[
\begin{align*}
\text{SELECT} & \quad \pi \text{Name} \\
& \quad \sigma \text{DeptId} = 'CS' \land \text{Semester} = 'F1994' \\
& \quad \text{FROM} \quad \text{Professor P, Teaching T} \\
& \quad \text{WHERE} \quad \text{P.Id = T.ProfId} \\
& \quad \quad \land \quad \text{P.DeptId} = 'CS' \land \text{T.Semester} = 'F1994'
\end{align*}
\]

π Name(σ DeptId = 'CS' ∧ Semester = 'F1994' (Professor × Id = ProfId Teaching))

Profesor Teaching

Black Nested Loop Join

Estimating Cost - Example 1

• **Join** - block-nested loops with 52 page buffer
  – 50 pages for input for Professor
  – 1 page for input for Teaching
  – 1 page as output page
• Scanning Professor (outer loop):
  – 200 page transfers, 4 iterations, 50 transfers each
• Finding matching rows in Teaching (inner loop):
  – 1000 page transfers for each iteration of outer loop
• Total cost of join = 200 + 4*1000 = 4200 page transfers

Estimating Cost - Example 1 (cont’d)

• **Selection and projection:**
  – scan rows of intermediate file
  – discard those that don’t satisfy selection
  – project on those that do
  – write result when output buffer is full
• Complete algorithm:
  – do join, write result to intermediate file on disk
  – read intermediate file, do select/projection, write final result
• **Problem:** unnecessary I/O
Pipelining

- **Solution:** use pipelining:
  - `join` and `select/project` act as coroutines, operate as producer/consumer sharing a buffer in main memory
    - When `join` fills buffer, `select/project` filters it and outputs result
    - Process is repeated until `select/project` has processed last output from `join`
  - Performing `select/project` adds no additional I/O cost

```
+----+-----------------+-------------------+---------------------+
<table>
<thead>
<tr>
<th></th>
<th>join</th>
<th>Intermediate result</th>
<th>select/project</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>buffer</td>
<td></td>
<td>output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>final result</td>
</tr>
</tbody>
</table>
```

Estimating Cost - Example 1 (cont’d)

- **Total cost:**
  \[ 4200 + \text{(cost of outputting final result)} \]
  - We will disregard the cost of outputting final result in comparing with other query evaluation strategies, since this will be same for all

```
\pi\text{Name(} \sigma\text{DeptId='CS'(Professor)) \times \sigma\text{Semester='F1994'(Teaching)}\]
```

Example 2

\[
\pi_{\text{Name}}(\sigma_{\text{DeptId='CS'}}(\text{Professor}) \times \sigma_{\text{Semester='F1994'}}(\text{Teaching}))
\]

Cost Example 2 -- *selection*

- Compute \(\sigma_{\text{DeptId='CS'}}(\text{Professor})\) using clustered, 2-level B\(^2\) tree on `DepId`.
  - 50 departments and 1000 professors; hence approximately 20 profs per department
  - These rows are in \(~4\) consecutive pages in `Professor`.
    - Cost = 4 (to get rows) + 2 (to search index) = 6
    - Keep resulting 4 pages in memory and pipe to next step
Cost Example 2 -- Join

- To access tuples in Teaching use clustered index on Semester
  - 2 I/Os to find the leaf block in the index
  - 250 I/Os to get the data blocks
    - 4 semesters, 250 pages match the given Semester
- Total Cost:
  - $2 + 4 + 2 + 250 = 258$
- 258 vs. 4200
  - Cost of output same as for Example 1

Comments On Optimization

- Fully pushed plans are not necessarily the “most” optimal
- Depending on the indexes available, partially pushed plans may be better than fully pushed plans
- So the Optimizer may come up with partially pushed plans and even plans which don’t use any of the available indexes
  - Depending upon whether the index is clustered or unclustered, and how selective is it
    - E.g., Index on an attribute ‘Sex’ may return half of the tuples. If such an index is unclustered, it may not get used

Topics

- Introduction
- Optimization Using Relational Algebra Equivalence
- A Quick Look at Implementation of Relational Operators
- Optimization Using Cost Estimates
- Query Plans in Oracle

(QUERY EXECUTION) Plan in Oracle

- EXPLAIN PLAN Utility
  - Shows the Plan Oracle will use to execute a query
- Two ways to use:
  1. In the current SQL Plus session, issue the statement:
     - SQL> SET AUTOTRACE ON EXPLAIN
     - Now Query Plan will be shown for every SQL statement executed in the SQL Plus session.
  2. Use ‘EXPLAIN PLAN’ command
     - Query Plan will be stored in a table
     - You will need to query that table and format the output
     - Con: More complicated to use than autotrace
     - Pro: The Optimizer does not execute the query. It merely states what would be done if the query were executed
Pre-requisites for Explain Plan

- The current User needs to have
  - A table (PLAN_TABLE) with required attributes
  - Can be created by running a script provided by Oracle
  - Script available at the course homepage
  - Privileges on the objects in the plan
    - More of an issue for views
    - E.g., You have privilege to SELECT from the view USER_TABLES. If you don't have privilege on the tables on
      which this view is defined, your query will succeed but the plan will not be shown

Plan_TABLE: Important Attributes

- operation: varchar2(30) - SQL op performed in the step
- options: varchar2(255) - Option used for SQL op (e.g., RANGE SCAN)
- object_name: varchar2(30) - Name of object in the step
- object_type: varchar2(30) - Attribute of object (e.g., UNIQUE)
- id: numeric - Id of parent step in plan
- parent_id: numeric - Id of parent step in plan
- position: numeric - Order of processing within parent
- cost: numeric - RELATIVE cost estimate for step
- cardinality: numeric - Estimated rows returned by step
- bytes: numeric - Estimated bytes returned by step

Example: Create & Load Table

CREATE TABLE my_emp (emp_id NUMBER, name VARCHAR2(100),
  address VARCHAR2(200));

-- a PL/SQL procedure to load up this table
CREATE OR REPLACE PROCEDURE my_emp_load(start_id INTEGER,
  end_id INTEGER) AS
BEGIN
  FOR i IN start_id..end_id LOOP
    INSERT INTO my_emp VALUES (i,
      MOD(i, 200) || 'this is a loooooooooooong name',
      'this is a loooooooooooooooooooong address' || i);
    IF MOD(i, 5000) = 0 THEN
      COMMIT;
    END IF;
  END LOOP;
END;
/
EXECUTE emp_load(1, 50000);

Explain Plan Without Stats

- Set autotrace on explain and then run a query to see the plan

SQL> SELECT name FROM my_emp WHERE name = 'no such name';
no rows selected

Execution Plan

0 SELECT STATEMENT Optimizer=CHOOSE
  1 0 TABLE ACCESS (FULL) OF 'MY_EMP'
How to Read a Plan?

- First: inside-out
- Second: top-to-bottom

What’s Missing in the Previous Plan?
- The Plan doesn’t have any estimates of size of the results.
- The reason: no statistics were available to the Query Optimizer.
- Generally, statistics need to be explicitly collected.
- One mechanism in Oracle: ANALYZE command
  - E.g., ANALYZE TABLE my_emp COMPUTE STATISTICS;
  - There are other mechanisms available as well
  - Statistics gathering has side effects
    - Resource consumption, locking, plan invalidation (i.e., the old plan may be discarded. Not necessarily desirable)

Cost Estimate with Statistics
- If we analyze the table and again run the query:
  SQL> ANALYZE TABLE my_emp COMPUTE STATISTICS;
  SQL> SELECT name FROM my_emp WHERE name = 'no such name';

Execution Plan
----------------------------------------------------------
0      SELECT STATEMENT Optimizer=CHOOSE (Cost=67 Card=250 Bytes=8250)
  1  0   TABLE ACCESS (FULL) OF 'MY_EMP' (Cost=67 Card=250 Bytes=8250)
• If we examine the data that was loaded:
  - Number of rows = 50,000
  - Number of distinct values for name = 200
  - Cardinality = 250 \* 50,000/200=250
  - Average size of name column = 33 bytes
  - Bytes = 33*250=8250
• Query Optimizer gets these values from USER_TABLES and USER_TAB_COL_STATISTICS
Explain Plan With Index

- If we create an index on NAME attribute, analyze the index and run the query again

```
Execution Plan
---------------------
0 SELECT STATEMENT Optimizer=CHOOSE (Cost=4 Card=250 Bytes=8250)
1 0 INDEX (RANGE SCAN) OF 'NAME_IDX' (NON-UNIQUE)
   (Cost=4 Card=250 Bytes=8250)
```

- The number of rows and the result size result in bytes are the same, but the cost is much lower
- The access path in this case is the index
- In fact the query has been answered by using just the index
  - No need to go to the table
  - Such plans are called ‘Index-only’ plans

A “Quick and Dirty” Performance Measure

- Explain plan give a wealth of information
- The time to execute a query is a quick and dirty measure of the query execution plan
  - SQL> SET TIMING ON;
  - Now the time to execute a query will be shown
    - This time includes the time to display the results on the screen
    - If you don’t want this time counted, an alternative is to embed the query in SELECT COUNT(*) FROM (…)

```
SQL> select count(*) from (select name from my_emp where name like '199%');
```

References

- Oracle Documentation
  - Oracle9.2 Database Performance Tuning Guide and Reference
    - Chapters 1-9 are relevant to Optimizer and Query Plans
- Online papers
  - Search for “explain plan”
- Books on Oracle SQL Tuning
  - “Oracle SQL High-Performance Tuning” Guy Harrison
- Tools for Developers and DBA are also available for most DBMS. Include GUI’s to show plans
  - E.g., Oracle Enterprise Manager