Query Processing and Query Optimizer

ACS 575: Database Systems

Instructor: Dr. Jin Soung Yoo, Professor
Department of Computer Science
Purdue University Fort Wayne

References

- W. Lemanhieu, et al., Principles of Database Management: The Practical Guide to Storing, Managing and Analyzing Big and Small Data, Ch 13.1.3
- Elmasri et al., Fundamentals of Database Systems, Ch 18, 19
- Ramakrishnan et al., Database Management Systems, Ch4, **12**, 14, 15)

Outline

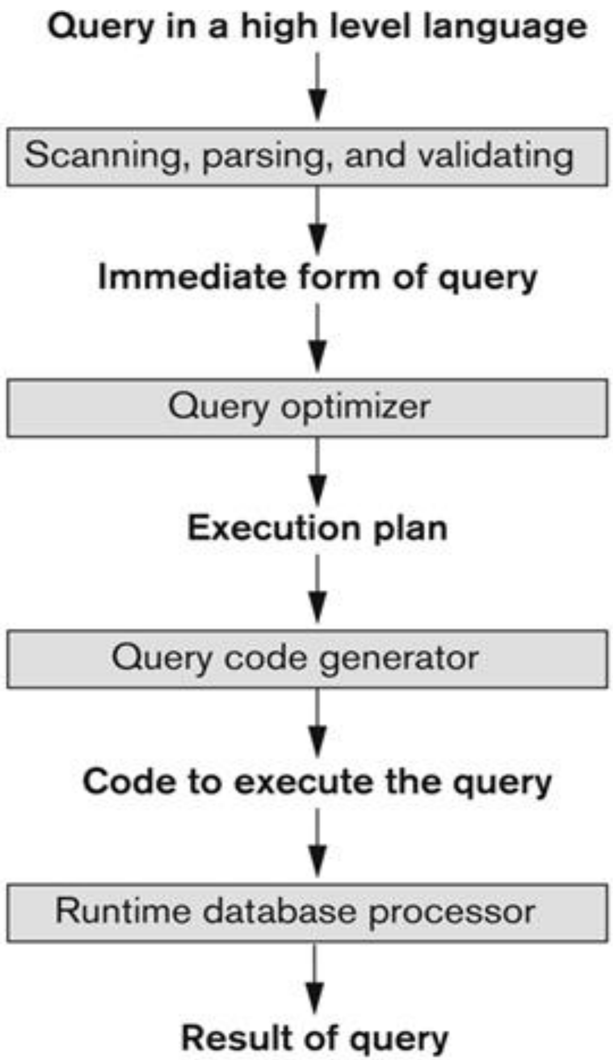
- Query Processing
- Database Access Methods
- Query Optimizer
- Optimizer Hints
- Querying Execution Plan

How DBMS Processes a Query

1. Scanner identifies query tokens
2. Parser checks the query syntax
3. Validation checks all attribute and relation names
4. Query trees (or query graph) are created
5. Execution strategy or query plan is devised
6. Code to execute the query is generated
7. The query is execute.

Query Processing Procedure

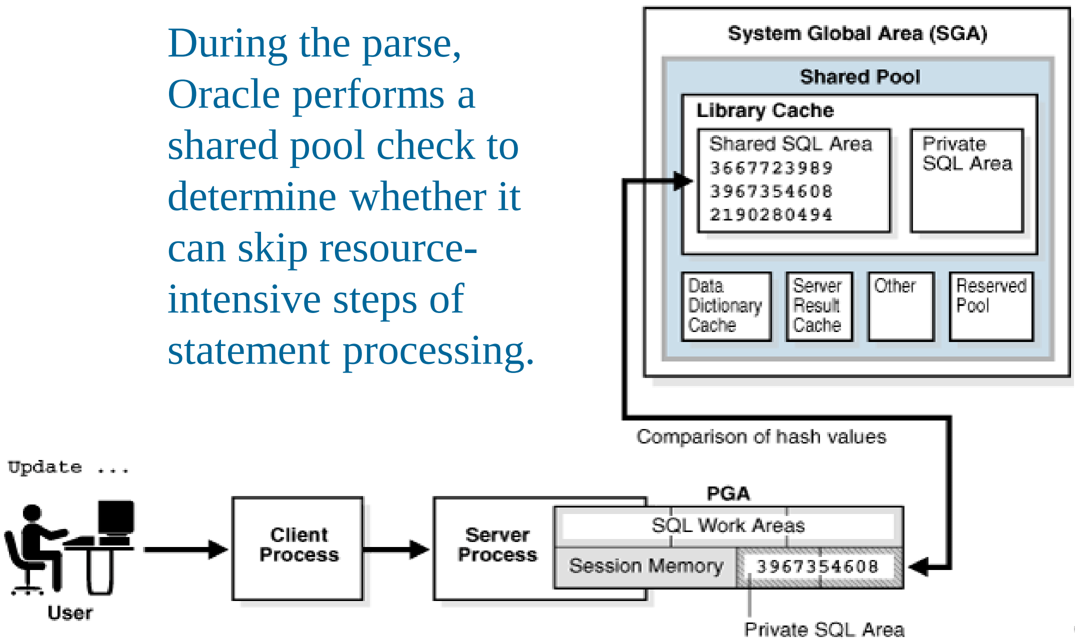
Typical steps when processing a high level query

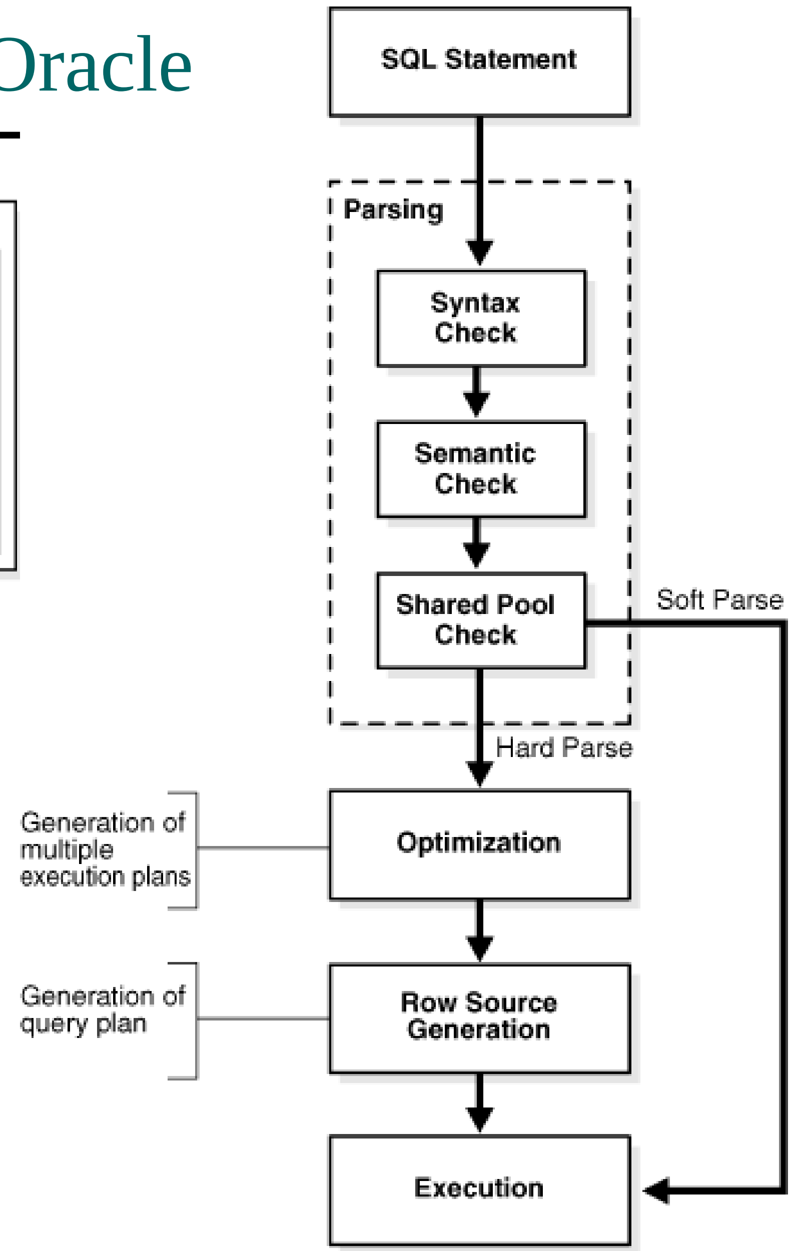


Query optimizer chooses a suitable execution strategy for processing a query.

Example: SQL Processing in Oracle

During the parse, Oracle performs a shared pool check to determine whether it can skip resource-intensive steps of statement processing.





Query Processor and Optimizer

- **Query processor** assists in execution of queries and consists of DML compiler, query parser, query rewriter, query optimizer and query executor
- **Query optimizer**
  - Planning a good execution strategy

Translating SQL Queries

- SQL queries are translated into a form of relational algebra
- A ***query block*** is the basic unit that can be translated into the algebraic operators and optimized.
  - A query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clause if these are part of the block.

SELECT LNAME "Last Name"
FROM EMPLOYEE
WHERE SALARY > C

$$\rho_{Lname \rightarrow Last\ Name}\pi_{\mathrm{LNAME}}(\sigma_{\mathrm{SALARY>C}}(\mathrm{EMPLOYEE}))$$

Translating SQL Queries

- Nested queries within a query are identified as separate query blocks.

```
SELECT  LNAME  "Last Name"
FROM    EMPLOYEE
WHERE   SALARY > ( SELECT  MAX (SALARY)
                   FROM    EMPLOYEE
                   WHERE   DNO = 5);
```

```
SELECT  LNAME  "Last Name"
FROM    EMPLOYEE
WHERE   SALARY > C
```

```
SELECT  MAX (SALARY)
FROM    EMPLOYEE
WHERE   DNO = 5
```

$\rho_{Lname \rightarrow Last\ Name}\pi_{\text{LNAME}}(\sigma_{\text{SALARY>C}}(\text{EMPLOYEE}))$

$\mathscr{F}_{\text{MAX SALARY}}(\sigma_{\text{DNO=5}}(\text{EMPLOYEE}))$

Reminder: Relational Algebra

- Basic operations:
  - *Selection* (σ): Selects a subset of tuples from relation.
  - *Projection* (π): Deletes unwanted attributes from relation.
  - *Cross-product* ($\times$): Allows us to combine two relations.
  - *Set-difference* ($-$) : Tuples in relation 1, but not in relation 2.
  - *Union* ($\cup$): Tuples in relation 1 or in relation 2.
  - *Intersection* ($\cap$)
- Additional operations:
  - *Join* ($\bowtie$)
  - *Renaming* (ρ)
  - Aggregation($\mathscr{F}$)

Query Processing

- Database systems have algorithms for each relational operation.
  - Algorithms for SELECTION operations
  - Algorithms for JOIN operations
  - Algorithms for PROJECT, SET, Aggregate operations, etc.
  - Algorithms for external sorting

Outline

- Query Processing
- ☞ **Database Access Methods**
- Query Optimizer
- Optimizer Hints
- Querying Execution Plan

Database Access Methods

- Different access paths exist to get to the same data, although the time to accomplish the retrieval task may vary greatly
- Database access methods for SELECTION operation
  - **Full Table Scan**
  - **Index Search** (with Atomic Search Key)
  - **Multicolumn Index Search**
  - **Index Only Access**

Full Table Scan

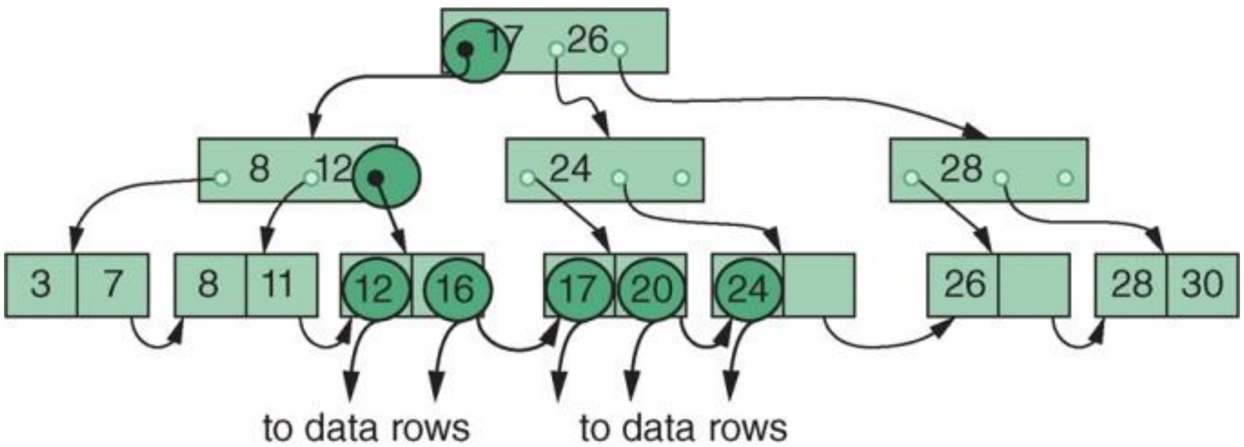
- If no index for a table query is available then need to <u>linearly search entire data pages (disk pages) of the table.</u>
- For very small tables, or for queries that require nearly all of a table's tuples anyway, full table scan might be more efficient
- The higher query's FF(**Filter Factor**), the less efficient a full table scan will be because this method needs to access almost all the data pages.
  - The FF of a query predicate represents the fraction of the total number of rows that is expected to satisfy the predicate.
- The larger the table, the less efficient a full table scan will be

Index Search (with Atomic Search Key)

- Single query predicate where query involves <u>search key with only single attribute type</u>

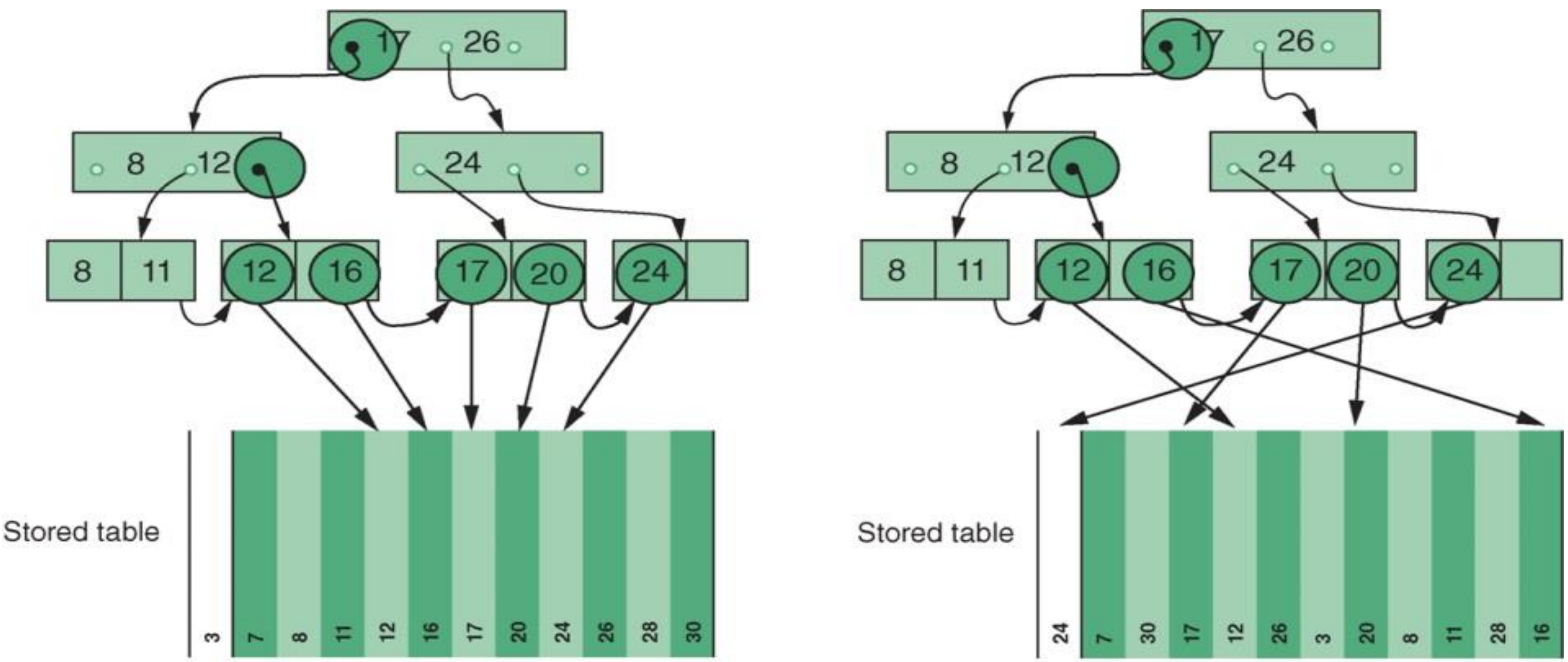
```
SELECT *
FROM MY_TABLE
WHERE MY_KEY >= 12 AND MY_KEY <= 24
```

- **Tree index** search using B^+-tree



Index Search (with Atomic Search Key)

- As to range queries, primary or clustered index is even more efficient than secondary index (sba (sequence block access) vs. rba (random block access))



Primary index   Secondary index

Multicolumn Index Search

- If search key is composite and indexed attribute types <u>are same as</u> attribute types in search key, then multicolumn index allows filtering out and retrieving only data rows that satisfy query
- A **tree index also** *matches* (a conjunction of) terms that <u>involve only attributes in a *prefix* of the search key.</u>
  - E.g., Tree index on <*a, b, c*> **matches** the selection *a*=*5 AND b*=*3*, and *a*=*5 AND b*>*6,* but **not marches** *b*=*3*.

Hashing Index Search

- If query involves only single search key value with equality operator, then **hashing** index would be efficient alternative
- A **hash index** *matches* (a conjunction of) terms that has a term *attribute = value* for every attribute in the search key of the index.
  - E.g., Hash index on <*a, b, c*> **matches**

    a=5 AND b=3 AND c=5;

    but it does **not match** *b=3, or a=5 AND b=3,*

    or a>5 AND b=3 AND c=5.

Index Only Access

- We might be 'lucky' such that query can be executed based solely on information in index
- Example

```
SELECT LASTNAME
FROM CUSTOMERTABLE
WHERE COUNTRY = 'U.K.' AND GENDER = 'M'
```

  - **Index only access** if there exists a multicolumn index, or a combination of single column indexes, over attribute types **LastName, Country** and **Gender**

- Note: the more attribute types are included in the index, the higher the negative performance impact of update queries!

Algorithms for SELECTION Operation

- Linear search (full table scan)
- Using a primary index or hash key to retrieve a single record:
- Using a secondary (B+-tree) index:
- Using a clustering index to retrieve multiple records:
- Using a secondary (B+-tree) index:

Example

- Relations
  - EMPLOYEE(SSN, NAME, SALARY, SEX, DNO)
  - DEPARTMENT(DNUMBER, DNAME)
  - WORKS_ON(ESSN, PNO)
  - PROJECT(PNO , PNAME, …)
- Indexes
  - Primary key (tree) indexes on key values,
  - Secondary (tree) index on SALARY,
  - Hash clustered index on ESSN and PNO
- Queries
  - (OP1): $\sigma_{\text{SSN='123456789'}}$ (EMPLOYEE)
  - (OP2): $\sigma_{\text{DNUMBER>5}}$(DEPARTMENT)
  - (OP3): $\sigma_{\text{DNO=5}}$(EMPLOYEE)
  - (OP4): $\sigma_{\text{DNO=5 AND SALARY>30000 AND SEX=F}}$(EMPLOYEE)
  - (OP5): $\sigma_{\text{ESSN=123456789 AND PNO=10}}$(WORKS_ON)

Algorithms for SELECTION Operation

- **Linear search (full table scan)**
  - Retrieve every record in the file, and test whether its attribute values satisfy the selection condition.
  - E.g., (OP3): $\sigma_{\text{DNO=5}}$(EMPLOYEE), no index on DNO

- **Using a primary index or hash key to retrieve a single record**
  - If the selection condition involves an equality comparison on a key attribute with a primary index (or a hash key), use the primary index (or the hash key) to retrieve the record.
  - E.g., (OP1): $\sigma_{\text{SSN='123456789'}}$ (EMPLOYEE)

    (OP5): $\sigma_{\text{ESSN=123456789 AND PNO=10}}$(WORKS_ON)

Algorithms for SELECT Operation (Contd.)

- **Using a primary index to retrieve multiple records**
  - If the comparison condition is $>$, $\geq$, $<$, or $\leq$ on a key field with a primary index, use the index to find the record satisfying the corresponding equality condition, then retrieve all subsequent records in the (ordered) file.
  - E.g., (OP2): $\sigma_{\text{DNUMBER}>5}(\text{DEPARTMENT})$
- **Using a clustering index to retrieve multiple records**
  - If the selection condition involves an equality (or range) comparison on a non-key attribute with a clustering index, use the clustering index to retrieve all the records satisfying the selection condition.
  - E.g., (OP5): $\sigma_{\text{ESSN}=123456789 \text{ AND } \text{PNO}=10}(\text{WORKS\_ON})$

Algorithms for SELECT Operation (Contd.)

- **Using a secondary (B+-tree) index**
  - On an equality comparison, this search method can be used to retrieve a single record if the indexing field has unique values (is a key) or to retrieve multiple records if the indexing field is not a key.
  - In addition, it can be used to retrieve records on conditions involving >,>=, <, or <=. (FOR RANGE QUERIES)
  - E.g., (OP2): $\sigma_{DNUMBER>5}$(DEPARTMENT)
  - $\sigma_{salary>70000}$(EMPLOYEE)

Algorithms for SELECT Operation (Contd.)

- Conjunctive selection:
  - If an attribute involved in any single simple condition in the conjunctive condition has an access path that permits the use of one of the methods S2 to S5, use that condition to retrieve the records and then check whether each retrieved record satisfies the remaining simple conditions in the conjunctive condition.
  - E.g., (OP4): $\sigma_{\text{DNO=5 AND SALARY>30000 AND SEX=F}}$(EMPLOYEE)

- Conjunctive selection using a composite (/multicolumn) index
  - If two or more attributes are involved in equality conditions in the conjunctive condition and a composite index (or hash structure) exists on the combined field, we can use the composite index directly.

JOIN Operation

- A **join query** between two tables specifies selection criteria that relate tuples from two tables to one another, according to a so-called join operator.
- It is one of the most time-consuming operations in an RDBMS.
- Inner Join: $\boldsymbol{R} \underset{r(a)\theta s(b)}{\bowtie} \boldsymbol{S}$, where the θ operator specifies the join condition, which is the criteria that determine which rows from table R are combined with which rows from table S.

  Equality search, e.g., r(a)=s(b) or an inequality search, e.g., r(s) $\geq$ s(b)

Table R

Employee	Payscale
Cooper	1
Gallup	2
O'Donnell	1
Smith	2

Table S

Payscale	Salary
1	10000
2	20000

$R \underset{r(payscale)=s(payscale)}{\bowtie} S$

Employee	Payscale	Salary
Cooper	1	10000
Gallup	2	20000
O'Donnell	1	10000
Smith	2	20000

Algorithms for JOIN Operation

- **Nested-loop join** (nested-block join)
- **Index-based nested-loop join**
- **Sort-merge join**
- **Hash Join**
- **Partition-hash join**

Algorithms for JOIN Operation

- **Nested-Loop Join**
  - One of the tables is denoted as the inner table and the other becomes the outer table.
  - For each row in the outer table, all rows of the inner table are retrieved and compared to the current row of the outer table.
  - If the join condition is satisfied, both rows are joined and put in an output buffer.
  - The inner table is traversed as many times as there are rows in the outer table.

Algorithms for JOIN Operation

- **Sort-Merge Join**
  - The tuples in both tables are first sorted according to the attribute types involved in the join condition.
  - Both tables are then traversed in this order, with the rows that satisfy the join condition being combined and put in an output buffer

Algorithms for PROJECT and Set Operations

- PROJECT operation
  - After projecting *R* on only the columns in the list of attributes, any duplicates are removed by treating the result strictly as a set of tuples
- Default for SQL queries
  - No elimination of duplicates from the query result
    - Duplicates eliminated only if the keyword DISTINCT is included
- Set operations
  - UNION
  - INTERSECTION
  - SET DIFFERENCE
  - CARTESIAN PRODUCT
- Set operations sometimes expensive to implement
  - Sort-merge technique
  - Hashing

Algorithms for Aggregate Operations

- Aggregate operators
  - MIN, MAX, COUNT, AVERAGE, SUM
  - Can be computed by a table scan or using an appropriate index
- AVERAGE or SUM
  - Index can be used if it is a dense index
  - Computation applied to the values in the index
  - Nondense index can be used if actual number of records associated with each index value is stored in each index entry
- COUNT
  - Number of values can be computed from the index

Outline

- Query Processing
- Database Access Methods
- **Query Optimizer**
  - Rule-based (/Heuristic) Optimizer
  - Cost-based Optimizer
- Optimizer Hints
- Querying Execution Plan

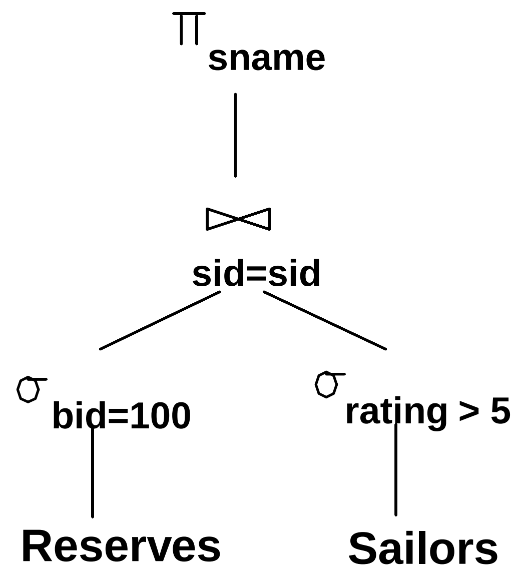
Query Optimizer

- SQL is a declarative query language in which a developer specifies ***which*** **data** are required, but **not *how* the data** are to be located and retrieved from the physical database files.
- Different access paths exist to get to the same data, although the time to accomplish the retrieval task may vary greatly
- The **query optimizer** optimizes the query, taking the current database state into account, as well as information in the catalog and available access structures such as indexes.
  - For each query, it is the responsibility of the *optimizer* to translate the different possible ways of resolving the query into different access plans and to select the plan with the highest estimated efficiency.
- The result of the query optimization procedure is a final access plan which is then handed over to the query executor for execution

Query Execution Plan

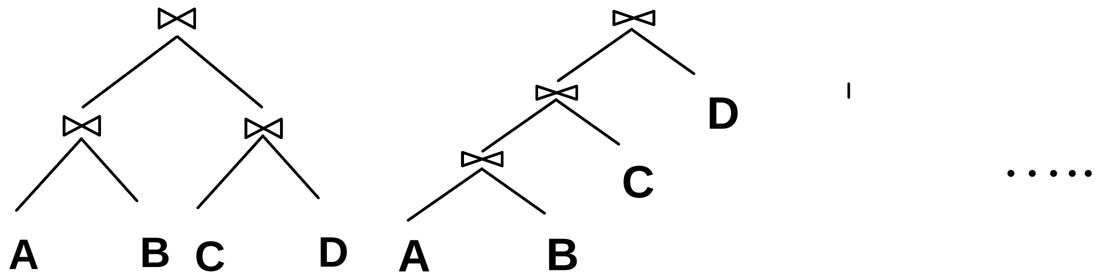
- Query optimization is the process of
  - choosing a suitable execution strategy for processing a query, and
  - generating groups of operations based on the access paths available on the files involved in the query.
- **Execution Plan**: A tree data structure of relational algebra operations, with choice of algorithm for each operation.
  - It represents the input relations of the query as leaf nodes, and the relational algebra operations as internal nodes.
- Example

```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating > 5;
```



Query Optimization

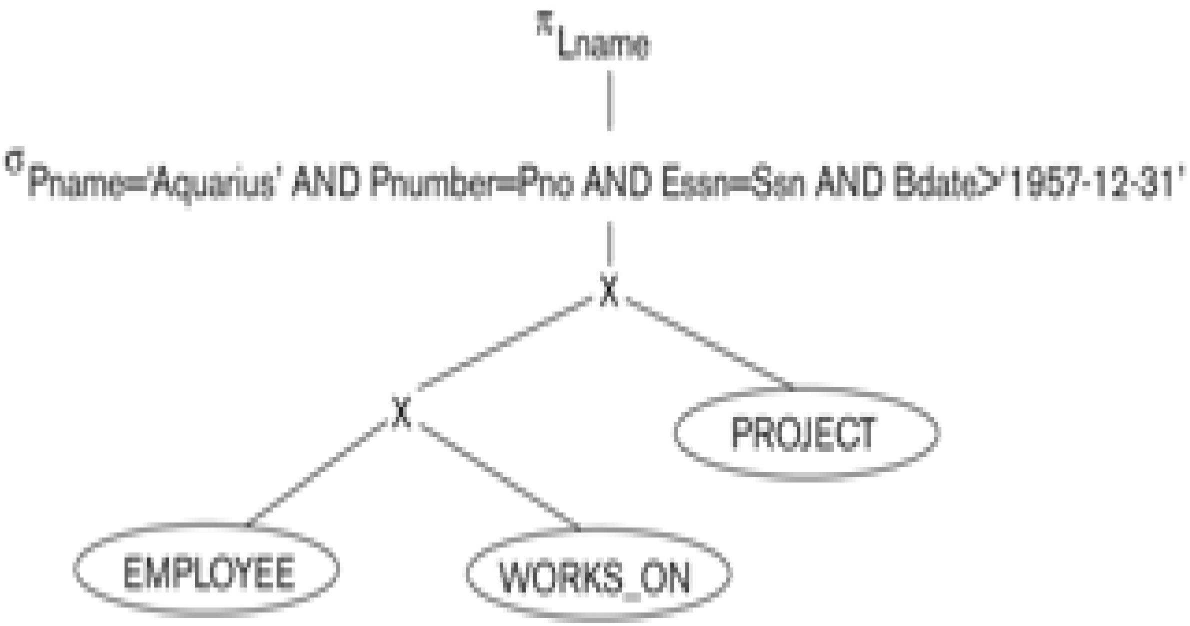
- For a given query, what plans are considered?
  - The same query could correspond to many different relational algebra expressions — and hence many different query trees.



  - *Goal of optimization*: To find more efficient plans that compute the same answer.
    - Search plan space for cheapest (estimated) plan.
  - Ideally: Want to find best plan. Practically: Avoid worst plans!

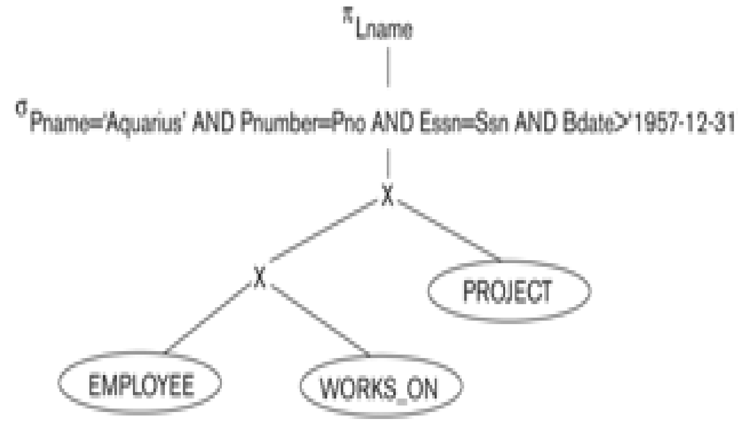
Example: Initial (Canonical) Query Tree

```
SELECT    LNAME
FROM      EMPLOYEE, WORKS_ON, PROJECT
WHERE     PNAME = 'AQUARIUS'
    AND   PNMUBER=PNO AND ESSN=SSN
    AND   BDATE > '1957-12-31';
```



Initial (Canonical) Query Tree (Cont.)

- By no means the worst plan!
  - Cross Product operation is expensive.



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS
Andy	C	Vile	222222202	21-JUN-44	1967 Jordan, Milwaukee, WI
Brad	C	Knight	111111103	13-FEB-68	176 Main St., Atlanta, GA
Evan	E	Wallis	222222200	16-JAN-58	134 Pelham, Milwaukee, WI
Josh	U	Zell	222222201	22-MAY-54	266 McGrady, Milwaukee, WI
Jared	D	James	111111100	10-OCT-66	123 Peachtree, Atlanta, GA
Justin	n	Mark	111111102	12-JAN-66	2342 May, Atlanta, GA
Jon	C	Jones	111111101	14-NOV-67	111 Allgood, Atlanta, GA
John	C	James	555555500	30-JUN-75	7676 Bloomington, Sacramento, CA
Alex	D	Freed	444444400	09-OCT-50	4333 Pillsbury, Milwaukee, WI
Ahmad	V	Jabbar	987987987	29-MAR-59	980 Dallas, Houston, TX
Joyce	A	English	453453453	31-JUL-62	5631 Rice, Houston, TX
Ramesh	K	Narayan	666884444	15-SEP-52	971 Fire Oak, Humble, TX
Alicia	J	Zelaya	999887777	19-JUL-58	3321 Castle, Spring, TX
John	B	Smith	123456789	09-JAN-55	731 Fondren, Houston, TX
Jennifer	S	Wallace	987654321	20-JUN-31	291 Berry, Bellaire, TX
Franklin	T	Wong	333445555	08-DEC-45	638 Voss, Houston, TX
James	E	Borg	888665555	10-NOV-27	450 Stone, Houston, TX
Tom	G	Brand	222222203	16-DEC-66	112 Third St, Milwaukee, WI
Jenny	F	Vos	222222204	11-NOV-67	263 Mayberry, Milwaukee, WI
Chris	A	Carter	222222205	21-MAR-60	565 Jordan, Milwaukee, WI
Kim	C	Grace	333333300	23-OCT-70	6677 Mills Ave, Sacramento, CA

WORKS_ON

ESSN	PNO	HOURS
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	null

Type of Query Optimizer

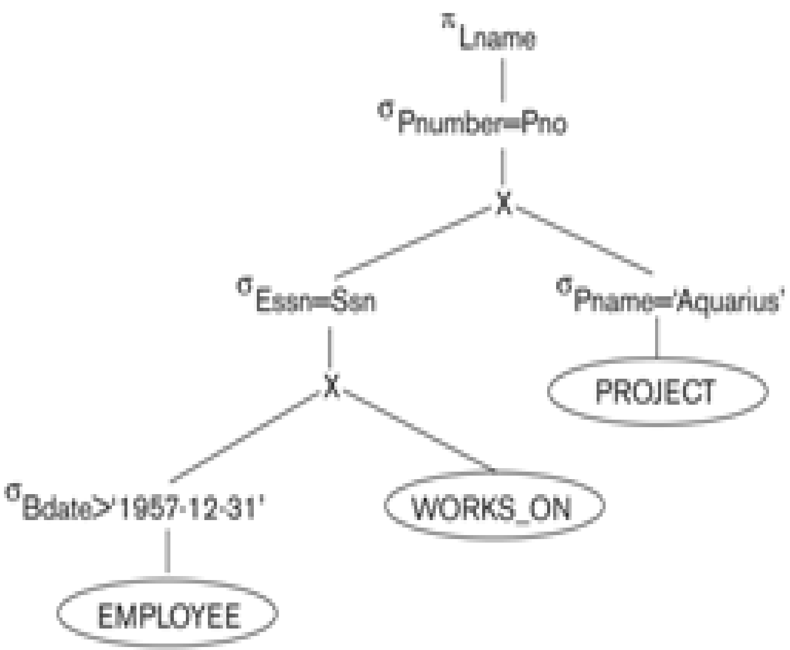
- Rule-based (/Heuristic) optimizer
- Cost-based optimizer

Heuristic Optimization of Query Tree

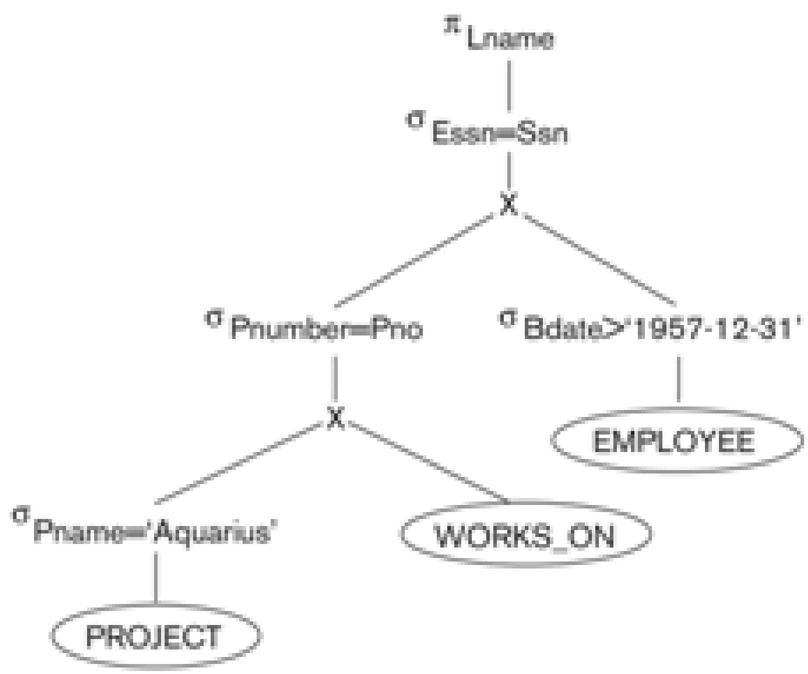
- **Heuristic optimizer** finds a final query tree that is efficient to execute, <u>based on some rules</u>**.**
- The main heuristic is <u>to apply first the operations that reduce the size of intermediate results.</u>
  - <u>Perform select operations as early as possible</u> to reduce the number of tuples, and <u>perform project operations as early as possible</u> to reduce the number of attributes.
    - This is done by moving select and project operations as far down the tree as possible.
  - <u>The select and join operations that are most restrictive should be executed before other similar operations</u>.
    - This is done by reordering the leaf nodes of the tree among themselves and adjusting the rest of the tree appropriately.

Alternative Query Plans

- Alternative plan 1
  - Moving SELECT operations down the query tree

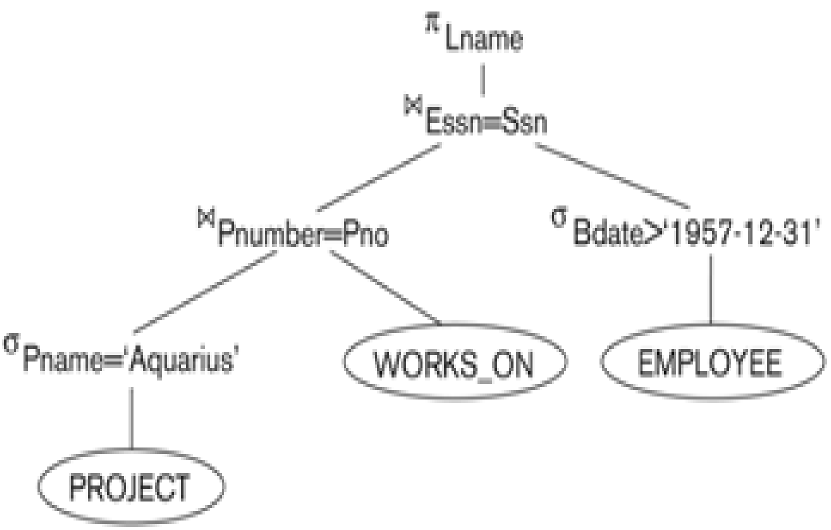


- Alternative plan 2
  - Applying the more restrictive SELECT operation first

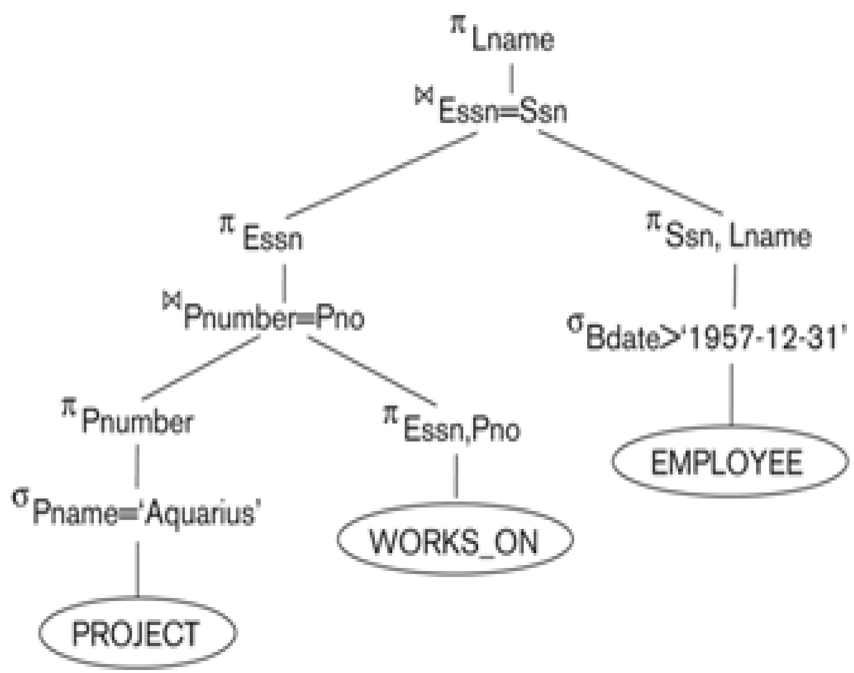


Alternative Query Plans

- Alternative plan 3
  - Replacing CARTESIAN PRODUCT and SELECT with JOIN operations



- **Final plan**
  - Moving PROJECT operations down the query tree



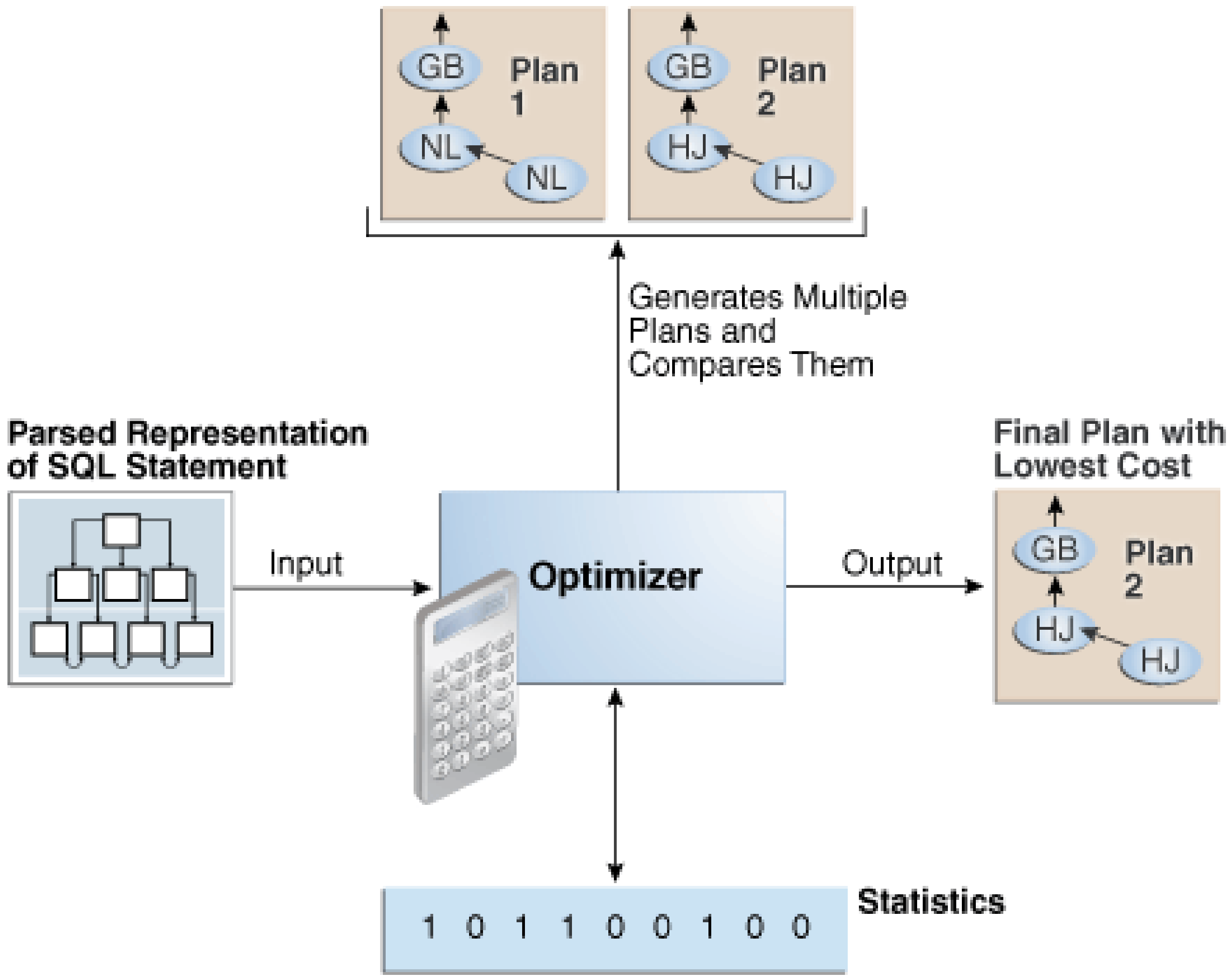
Outline

- Query Processing
- Database Access Methods
- Query Optimizer
  - Rule-based (/Heuristic) Optimizer
  - ☞ **Cost-based Optimizer**
- Optimizer Hints
- Querying Execution Plan

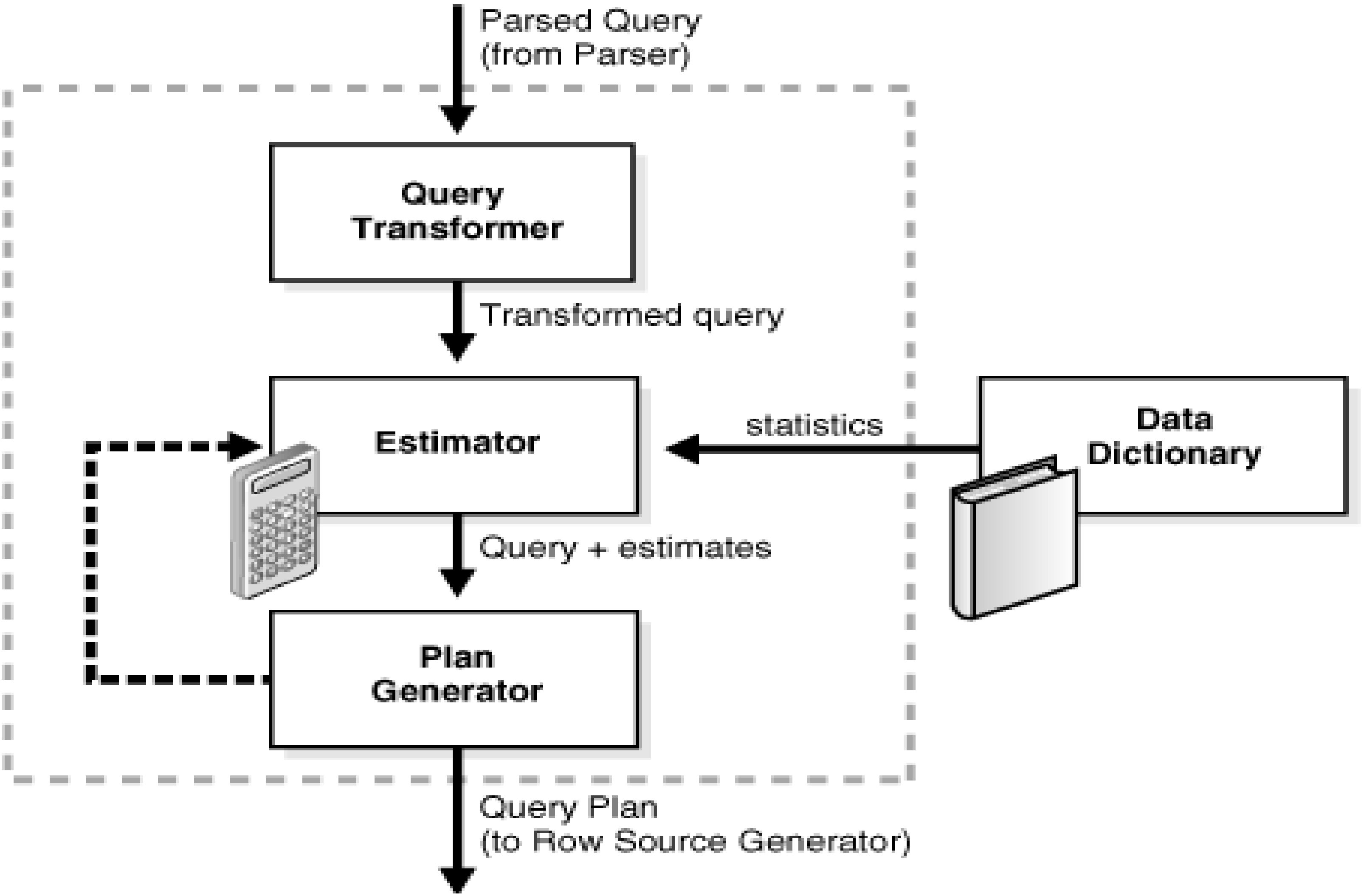
Cost-based Query Optimization

- **Cost-based optimizers** calculate optimal access plan according to set of built-in cost formulas as well as table(s) involved in query, available indexes, statistical properties of data in tables, etc.
- Estimate and compare the costs of executing a query using different execution strategies, and choose the strategy with the lowest cost estimate.

Query Processing by Cost-based Query Optimizer



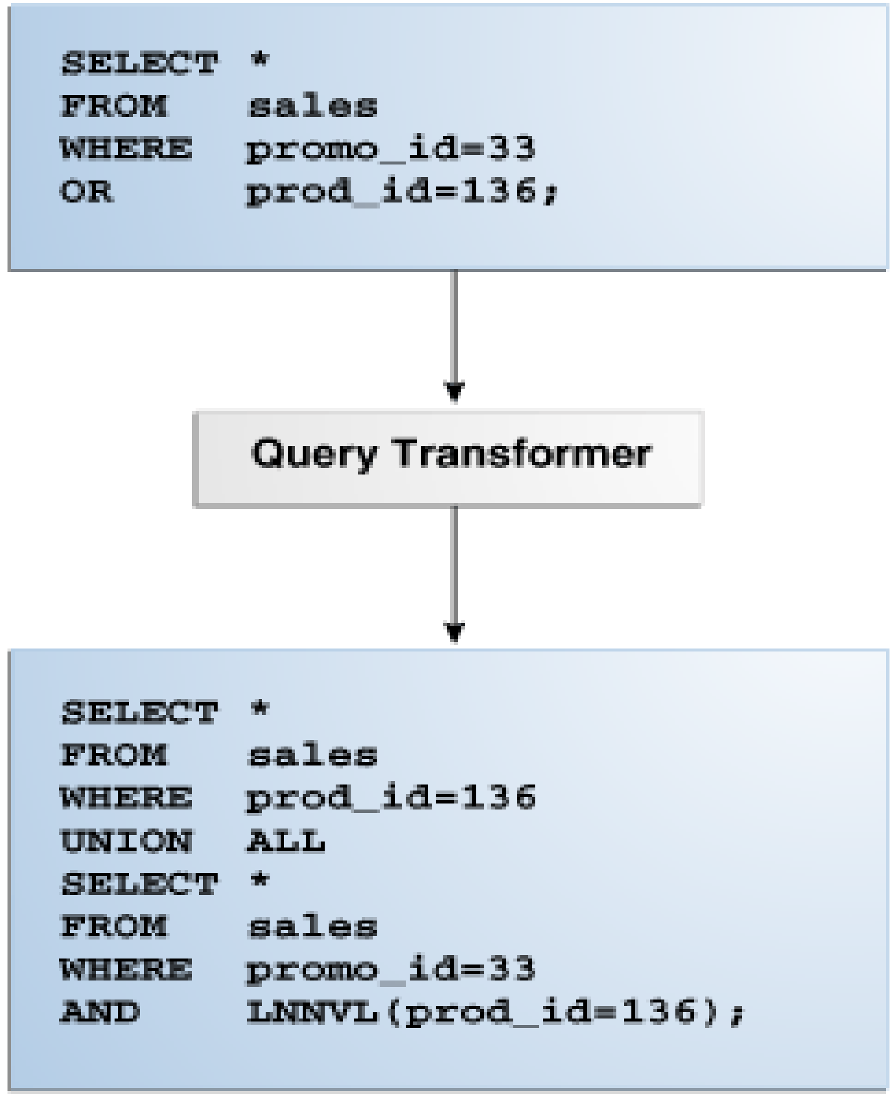
Optimizer Components



Query Transformer

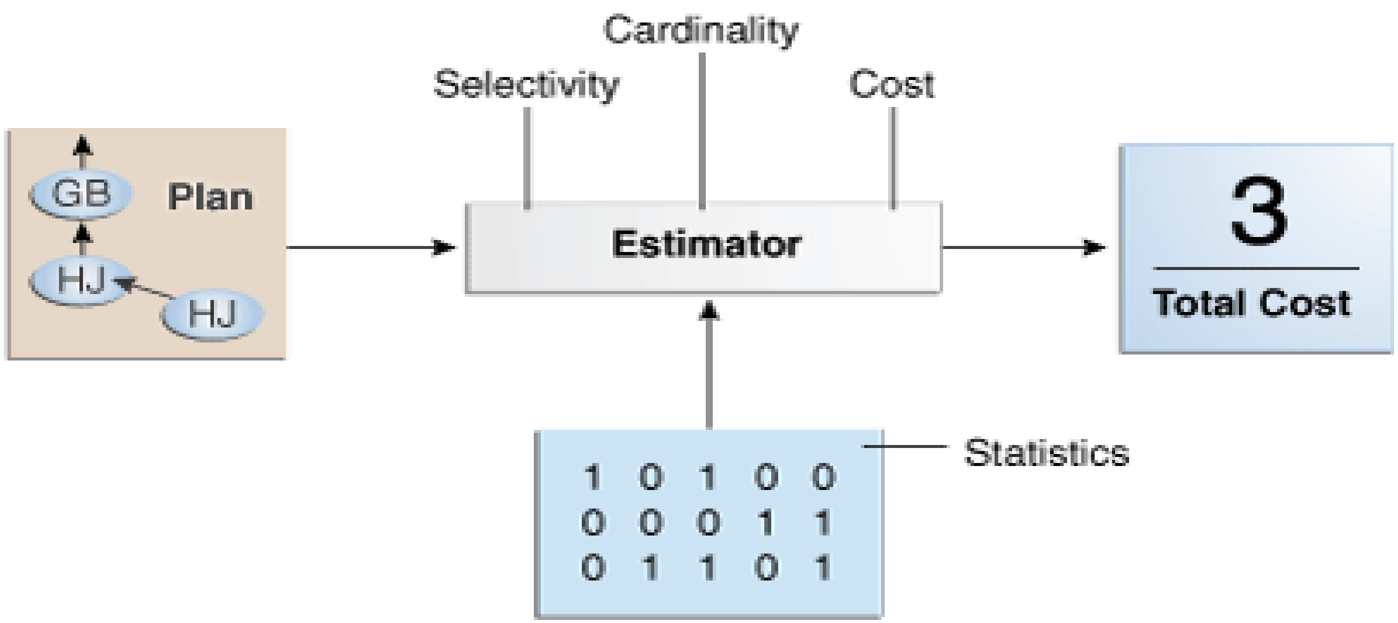
- The optimizer determines whether it is helpful to change the form of the query so that the optimizer can generate a better execution plan.

- An example with a query with OR operation



Estimator

- The optimizer estimates the cost of each plan based on statistics in the data dictionary.
- The estimator uses three different measures to determine the query execution cost: Selectivity, Cardinality, and Cost



Selectivity

- ***Selectivity*** (***Filtering Factor***) is a fraction rows in the row set that the query selects
- Selectivity is tied to a query predicate, such as `WHERE last_name='Smith',` or a combination of multiple predicates.
- Default estimate for FF_i is $1/NV_i$, with NV_i representing number of different values of attribute A_i
  - E.g., If the number of distinct values of `last_name` is 150, FF is 1/150=0.06
- A predicate becomes more selective as the selectivity (FF) value approaches 0 and less selective (or more unselective) as the value approaches 1. here, 0 meaning no rows and 1 meaning all rows.
  - Note that selectivity is an internal calculation that is not visible in execution plans.

Cardinality

- ***Cardinality*** is the number of rows returned by each operation in an execution plan.
  - E.g., If the optimizer estimate for the number of rows returned by a full table scan is 100, then the cardinality estimate for this operation is 100.
- The estimator can derive cardinality from the table statistics collected by DBMS_STATS, or derive it after accounting for effects from predicates (filter, join, and so on), DISTINCT or GROUP BY operations, and so on.
  - E.g., `SELECT… FROM employees WHERE salary=‘10200’.` If the employee table contains 107 rows and the number of distinct values in salary is 58, the cardinality of the result set is 107/58=1.84 (around 2)

Cost

- ***Cost*** measure represents units of work or resource used. The query optimizer uses disk I/O, CPU usage, and memory usage as units of work.

Data Dictionary (/ System Catalog)

- DBMS maintains following data in catalog
  - Table related data
    - number of rows, number of disk blocks occupied by table, number of overflow records associated with table
  - Column related data
    - number of different column values, distribution of column values
  - Index related data
    - number of different values for indexed search keys and for individual attribute types of composite search keys, number of disk blocks occupied by index, index type (primary/clustered or secondary)
  - and many others

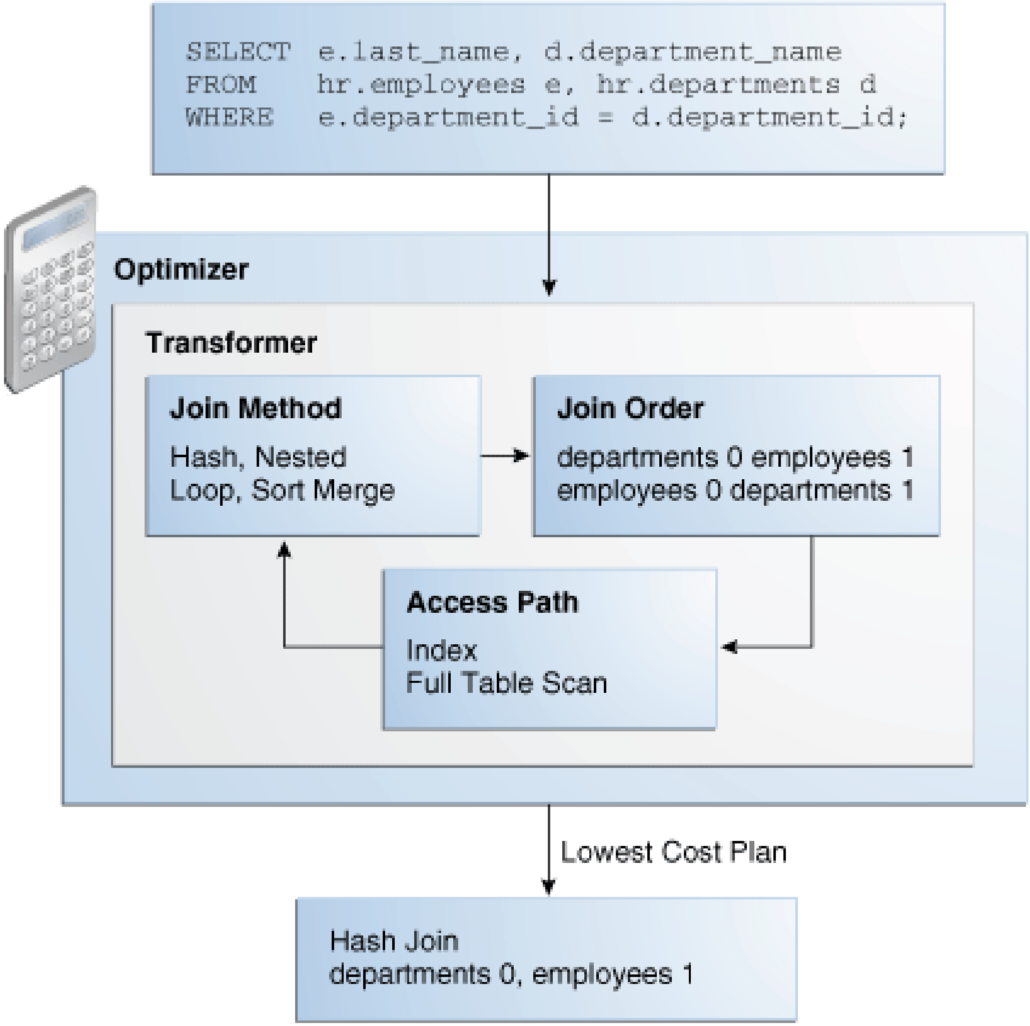
Refer to System Catalog slide.

Example of Information in Data Dictionary

- Catalog data about the size of a file
  - number of records (tuples) (*r*),
  - record size (*R*),
  - number of blocks (*b*)
  - blocking factor (*bfr*)
- Catalog data about indexes and indexing attributes of a file
  - Number of levels (*x*) of each multilevel index
  - Number of first-level index blocks (*bI1*)
  - Number of distinct values (*d*) of an attribute
  - Selectivity (*sl*) of an attribute
  - Selection cardinality (*s*) of an attribute. ($s = sl * r$)
- And so on…

Plan Generator

- The optimizer compares the costs of plans and chooses the lowest-cost plan, known as the execution plan, to pass to the row source generator.



Outline

- Query Processing
- Database Access Methods
- Query Optimizer
- ☞ **Optimizer Hints**
- Querying Execution Plan

Optimizer Hints

- **Optimizer hints** are special instructions for the optimizer that are embedded inside the SQL statement.
- The idea is that optimizer may not always choose the best execution plan, and an application developer might know more information about the data and application.
- Application developers could also specify "*hints*" to the query optimizer to override the default query optimization

Using Hints

- In each statement, the hint goes directly after the Select, Delete, or Update keyword
- Hints are placed in the /*+ */ tag, where the hint goes after the + sign
  - E.g., SELECT /*+ ALL_ROWS */ From…
- Types of Hints:
  - Hints for Optimization Approaches and Goals
  - Hints for Access Paths
  - Hints for Query Transformations
  - Hints for Join Orders
  - Hints for Join Operations
  - etc.

Example: Approach Hints

- ALL_ROWS
  - Minimizes total resource consumption. Results will be returned only after all processing has been completed

    SELECT /*+ ALL_ROW */ *

    FROM product

    where p_qoh <10;

- FIRST_ROWS
  - Minimized response time, or minimal resource usage to return the first *n* rows.

    SELECT /*+ FIRST_ROWS */ *

    FROM product

    where p_qoh <10;

Example: Access Hints

- FULL (table)
  - Chooses a full table scan for the table, even if there is an index available

    SELECT /*+ FULL(s)*/ id, name

    FROM student s

    WHERE sex =‘m’;

- INDEX (table index [table index] …])
  - Chooses an Index scan for the table.

    SELECT /*+ INDEX(s sex_index) */ id, name

    FROM student s

    WHERE sex =‘f’;

- Reference:

http://download.oracle.com/docs/cd/B10501_01/server.920/a96533/hintsref.htm

Outline

- Query Processing
- Database Access Methods
- Query Optimizer
- Optimizer Hints
- **Querying Execution Plan**

How to See the Execution Plan

- DMBS stores the query execution plan in a database and we can see the query plan of a SQL statement.
- Oracle store the query plan in PLAN_TABLE.
- There are several ways to see the plan table.

1) Populate PLAN_TABLE with execution plan of a SQL statement.

```
EXPLAIN PLAN
SET STATEMENT_ID = '<some-name>'
FOR <select statement to be analyzed>;
```

Example:

```
EXPLAIN PLAN
SET STATEMENT_ID = 'Q1'
FOR
Select * from employee where dno=8;
```

2) Query the PLAN_TABLE

```
SELECT LPAD(' ', 2*LEVEL)||OPERATION||' '||OPTIONS||' '||OBJECT_NAME Query_Plan
FROM PLAN_TABLE
CONNECT BY PRIOR ID = PARENT_ID and STATEMENT_ID = '<some-name>'
START WITH ID=1 and STATEMENT_ID = '<some-name>'
ORDER BY ID;
```

How to See the Execution Plan using Developer Tool

- The easiest way is to see the plan result in Oracle SQL Developer Tool.
- Use “Explain” tab