Steps in Query Processing

1. Translation
   - check SQL syntax
   - check existence of relations and attributes
   - replace views by their definitions
   - generate internal query representation

2. Optimization
   - consider alternative plans for processing the query
   - select an efficient plan

3. Processing
   - execute the plan

4. Data Delivery
Example

```
select  DeptNo, Deptname, count(*), sum(Salary)
from    Employee, Department
where   WorkDept = DeptNo and DeptNo like 'D%'
group by DeptNo, Deptname
having  sum(Salary) > 1000000
```
An Execution Plan

1. Scan the Employee table, select all tuples for which WorkDept starts with 'D', call the result $R_1$.

2. Join $R_1$ and Department, eliminate attributes other than DeptNo, Deptname, and Salary. Call the result $R_2$. This may involve:
   - sorting $R_1$ on WorkDept
   - sorting Department on Deptno
   - joining the two sorted relations to produce $R_2$

3. Group the tuples of $R_2$. Call the result $R_3$. This may involve:
   - sorting $R_2$ on DeptNo and Deptname
   - group tuples with identical values of DeptNo and Deptname
   - count tuples in each group, and add their Salaries

4. Scan $R_3$, select all tuples with $\text{sum}(\text{Salary}) > 1000000$
Pictorial Access Plan

Select
(Sum(Salary) > 100)

Grouping
(DeptNo, Deptname)

Project
(DeptNo, Deptname, Salary)

Join
(DeptNo = WorkDept)

Select
(DeptNo LIKE 'D%')

Department

Employee
Pipelined Plans and Iterators

• In a pipelined plan, each tuples stream from one operator to another.

• Pipelining allows for parallel execution of operators, and avoids unnecessary *materialization* of intermediate results. (Sometimes materialization may be necessary...)

• Iterators are a common model for plan operators:
  – every operator is an iterator
  – an iterator provides the following interface: *Open*, *GetNext*, and *Close*
  – each iterator implements its interface, using calls to the interface functions of its child (or children)
DB2 Access Plan

FILTER (having)

| GRPBY (deptno, deptname)
| MSJOIN

/ \
TBSCAN FILTER
| |
(deptno) SORT TBSCAN
| |
(like D%) TBSCAN SORT (workdept)
| |
DEPARTMENT TBSCAN (like D%)
| |
EMPLOYEE
DB2 Access Plan with Index

FILTER
  |
GRPBY
  |
TBSCAN
  |
SORT
  |
NLJOIN

/   \
TBSCAN FETCH
  |
EMPLOYEE IXSCAN DEPARTMENT
  |
DEPTNOIND
Some Basic Query Processing Operations

- Data Access and Filtering
  - Index scans
  - Table scans

- Projection

- Joining
  - nested loop join
  - hash join
  - sort-merge join
  - and others . . .

- Sorting

- Grouping and Duplicate Elimination
  - by sorting
  - by hashing
Joining Relations

```sql
select DeptName, LastName
from Department, Employee
where DeptNo = WorkDept
```

Conceptually, a nested-loop join works like this:

```python
foreach tuple d in Department do
    foreach tuple e in Employee do
        if d.DeptNo = e.WorkDept then
            output d, e
        end
    end
end
```
Block Nested Loop Join

```
select  DeptName, LastName
from    Department, Employee
where   DeptNo = WorkDept

Process outer relation a chunk at a time

foreach chunk C of Department
    foreach tuple e in Employee do
        foreach tuple d in C
            if d.DeptNo = e.WorkDept then
                output d, e
            end
        end
    end
end
```
Other Techniques for Join

- If there is an index on the WorkDept attribute of the Employee relation, an **index join** can be used:

  ```
  foreach tuple d in Department do
      use the index to find Employee tuples where d.DeptNo = WorkDept
      for each such tuple e
          output d,e
  end
  ```

- Examples of other join techniques:
  - **Sort-Merge Join**: sort the tuples of Employee on WorkDept and the tuples of Department of DeptNo, then merge the sorted relations.
  - **Hash Join**: assign each tuple of Employee and of Department to a “bucket” by applying a hash function to its WorkDept (DeptNo) value. Within each bucket, look for Employee/Department tuple pairs for which WorkDept = DeptNo.
Hash Join Example

- **Hash Join Operator**
- **Result**: 6, 1, 8, 1, 2, 8, 3, 3, 4, 6, 6, 8, 2, 8, 9, 3
- **Disk**: 8, 2, 2, 3, 4, 5, 5, 6, 5, 7, 7, 2, 3, 7, 8, 5
- **Memory**: 6, 1, 8, 1, 2, 8, 3, 3, 4, 6, 6, 8, 2, 8, 9, 3

**Diagram**: A diagram illustrating the hash join process with two sets (outer and inner) and their combined result.
Hash Join Example (cont’d)

Hash Join Operator

result

memory

outer

inner

disk

6,1,8,1,2,8,3,3,4,6,6,8,2,8,9,3

8,2,2,3,4,5,5,6,5,7,7,2,3,7,8,5

8,4,8
Hash Join Example (cont’d)

(8,8), (8,8), (8,8), (8,8), (4,4), (8,8), (8,8), (8,8), (8,8)

Hash Join Operator

result

disk

5,5,5,5
2,2,6,2
3,7,7,3,7
1,1,9
6,2,6,6,2
3,3,3

memory

6,1,8,1,2,8,3,3,4,6,6,8,2,8,9,3

outer

inner
Hash Join Example (cont’d)

Hash Join Operator

memory

result

disk

outer

inner

5,5,5,5
2,2,6,2
3,7,7,3,7
1,1,9
6,2,6,6,2
3,3,3
Hash Join Example (cont’d)

Hash Join Operator

memory

result

outer

inner

disk

2,2,6,2
3,7,7,3,7
1,1,9
6,2,6,6,2
3,3,3
Hash Join Example (cont’d)

(6,6),(2,2),(2,2),(2,2),(6,6),(6,6),(2,2),(2,2),(2,2)

Result

Hash Join Operator

Memory

Outer

Inner

Disk

2,2,6,2
3,7,7,3,7
6,2,6,6,2
3,3,3
Hash Join Example (cont’d)

(3,3),(3,3),(3,3),(3,3),(3,3),(3,3)

Hash Join Operator

result

memory

outer

inner

disk

3,7,7,3,7

3,3,3
External Merge Sort: Run Formation
External Merge Sort: Run Formation (cont’d)

Diagram showing the process of external merge sort with disk and memory components.
External Merge Sort: Run Formation (cont’d)
External Merge Sort: Merging Runs

disk

memory

disk
Summary

- A plan describes how a query is executed, including:
  - the sequence of basic operations (select, project, join, sort, etc.) used to process the query
  - how each operation will be implemented, e.g., which join method will be used, which indices will be used to perform a selection.