Query Processing: A Systems View

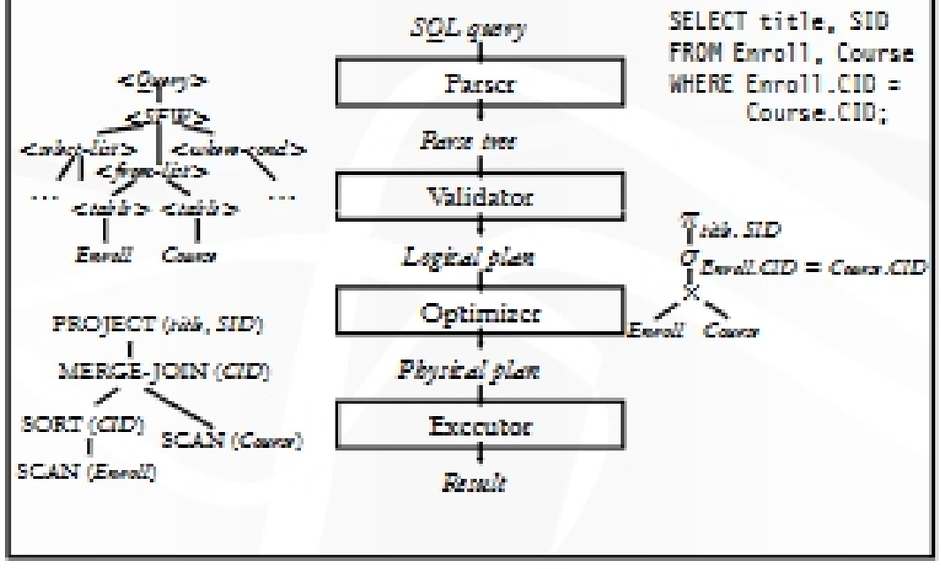
CPS 116

Introduction to Database Systems

Announcements (November 13)

- Homework #3 sample solution available
- Homework #4 due in 1½ weeks

A query's trip through the DBMS

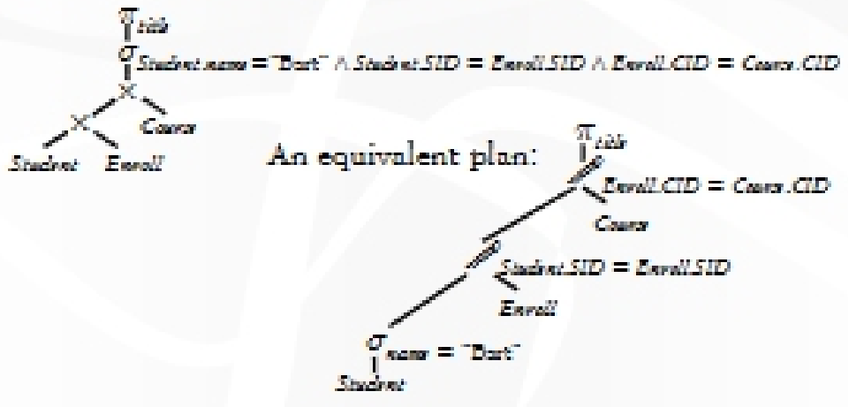


Parsing and validation

- Parser: SQL → parse tree
  - Good old lex & yacc will do
  - Detect and reject syntax errors
- Validator: parse tree → logical plan
  - Detect and reject semantic errors
    - Nonexistent tables/views/columns?
    - Insufficient access privileges?
    - Type mismatches?
      - Example: `AVG(name).name + GPA.Student UNION Enroll`
  - Also
    - Expand *
    - Expand view definitions
  - Information required for semantic checking is found in system catalog (contains all schema information)

Logical plan

- Nodes are logical operators (often relational algebra operators)
- There are many equivalent logical plans

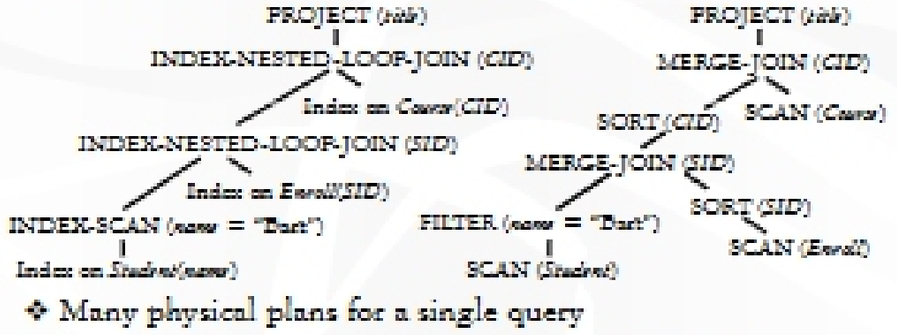


Physical (execution) plan

- A complex query may involve multiple tables and various query processing algorithms
  - E.g., table scan, index nested-loop join, sort-merge join, hash-based duplicate elimination…
- A physical plan for a query tells the DBMS query processor how to execute the query
  - A tree of physical plan operators
  - Each operator implements a query processing algorithm
  - Each operator accepts a number of input tables/streams and produces a single output table/stream

Examples of physical plans

```
SELECT Course.title
FROM Student, Enroll, Course
WHERE Student.name = 'Bart'
AND Student.SID = Enroll.SID AND Enroll.CID = Course.CID;
```



- Many physical plans for a single query
  - Equivalent results, but different costs and assumptions!
  - ☛ DBMS query optimizer picks the "best" possible physical plan

Physical plan execution

- How are intermediate results passed from child operators to parent operators?
  - Temporary files
    - Compute the tree bottom-up
    - Children write intermediate results to temporary files
    - Parents read temporary files
  - Iterators
    - Do not materialize intermediate results
    - Children pipeline their results to parents

Iterator interface

- Every physical operator maintains its own execution state and implements the following methods:
  - `open()`: Initialize state and get ready for processing
  - `getNext()`: Return the next tuple in the result (or a null pointer if there are no more tuples); adjust state to allow subsequent tuples to be obtained
  - `close()`: Clean up