CLOUDERA

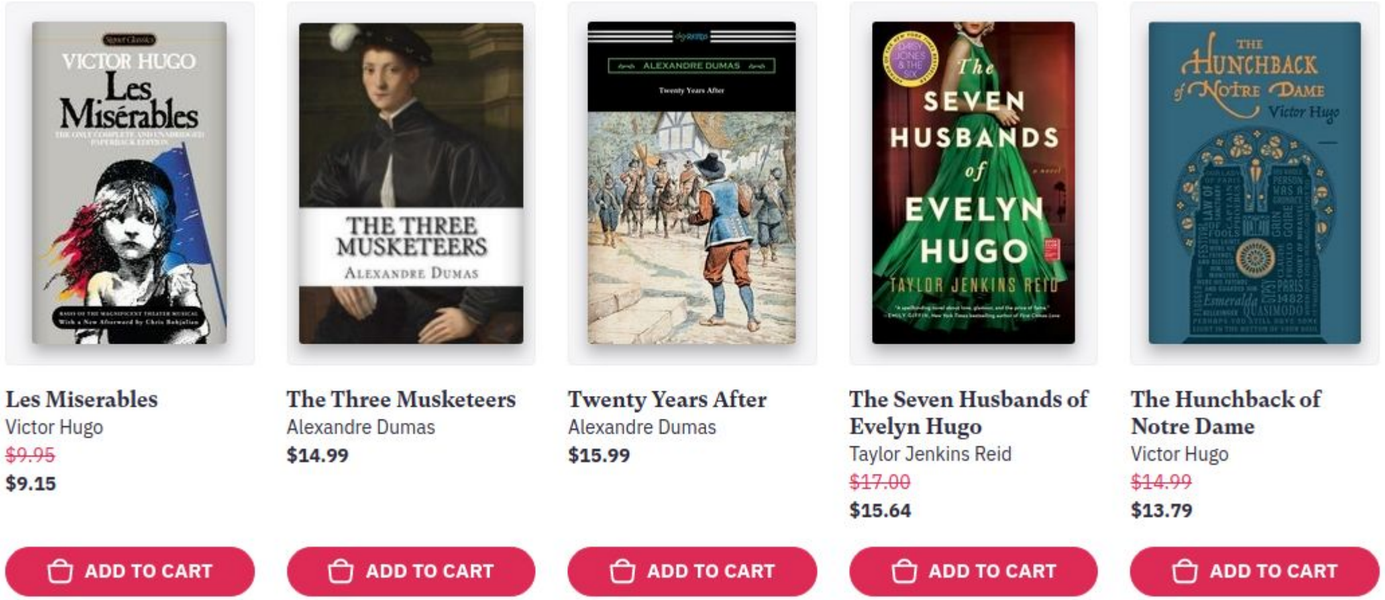
An introduction to query processing & Apache Calcite

by Stamatis Zampetakis

@Calcite Online Meetup

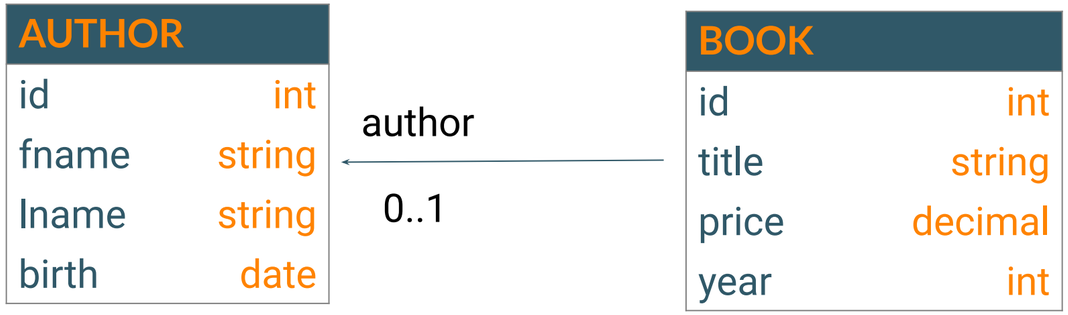
January 20, 2021

Motivation: Data views

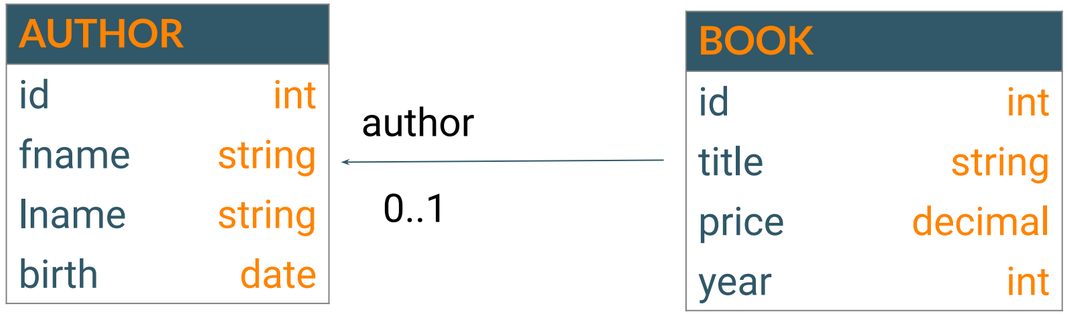


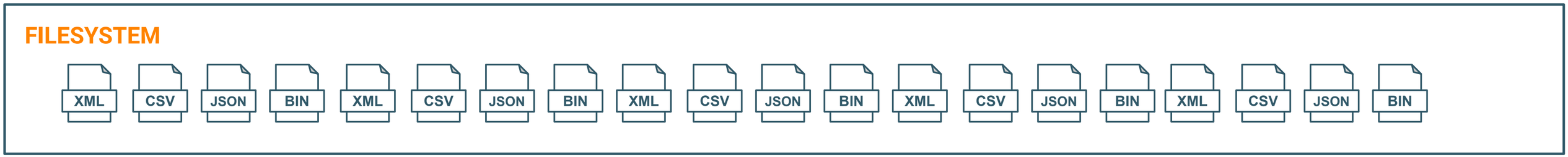
1. Retrieve books and authors
2. Display image, title, price of the book along with firstname & lastname of the author
3. Sort the books based on their id (price or something else)
4. Show results in groups of five

What, where, how data are stored?



What, where, how data are stored?





What, where, how data are stored?

AUTHOR	
id	int
fname	string
lname	string
birth	date

author

0..1

BOOK	
id	int
title	string
price	decimal
year	int

APACHE DRILL

APACHE Spark™

ORACLE DATABASE

cassandra

elasticsearch

HIVE

neo4j

hazelcast

Google BigQuery

APACHE HBASE

Lucene

PostgreSQL

redis

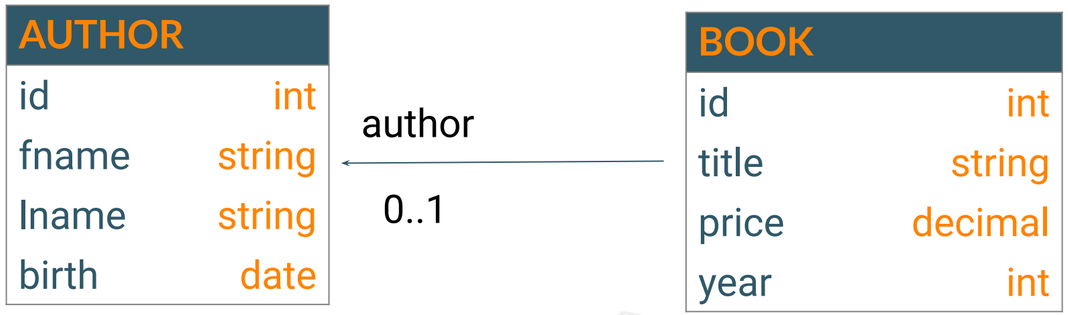
Solr

mongoDB®

FILESYSTEM

XML CSV JSON BIN XML CSV JSON BIN XML CSV JSON BIN XML CSV JSON BIN XML CSV JSON BIN

Many DBMS, different APIs

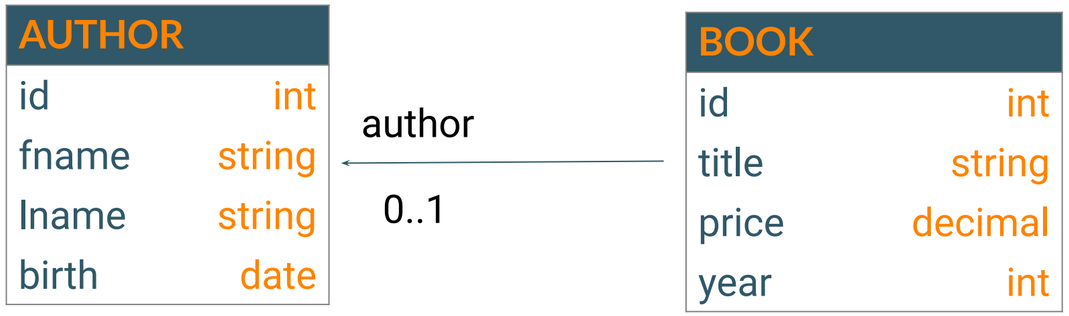


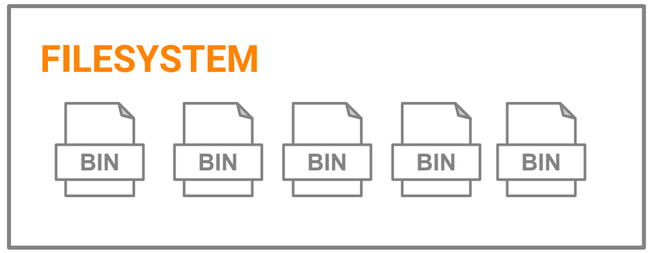
360+ DBMS

FILESYSTEM

XML CSV JSON BIN XML CSV JSON BIN XML CSV JSON BIN XML CSV JSON BIN XML CSV JSON BIN

In-house usecase specific system





- ★ Java based in-memory datastore
- ★ Ultra compact memory/disk format
- ★ High speed serialization/deserialization protocol

Naive implementation

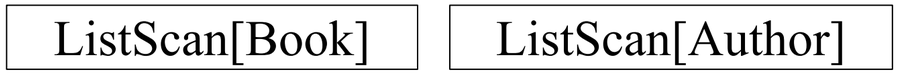
```
List<Tuple> buildMainView(List<Book> books, List<Author> authors, int limit)
```

Naive implementation

```
List<Tuple> buildMainView(List<Book> books, List<Author> authors, int limit)
{
   List<Tuple> result = new ArrayList<>();
   for (Book b : books)
   {









   }


}
```

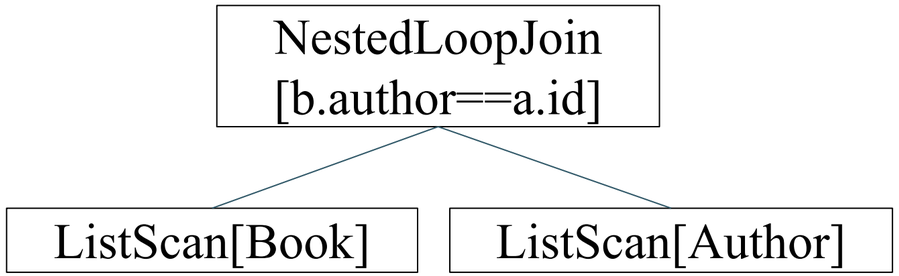
Naive implementation

```
List<Tuple> buildMainView(List<Book> books, List<Author> authors, int limit)
{
   List<Tuple> result = new ArrayList<>();
   for (Book b : books)
   {
      boolean hasAuthor = false;
      for (Author a : authors)



   }


}
```



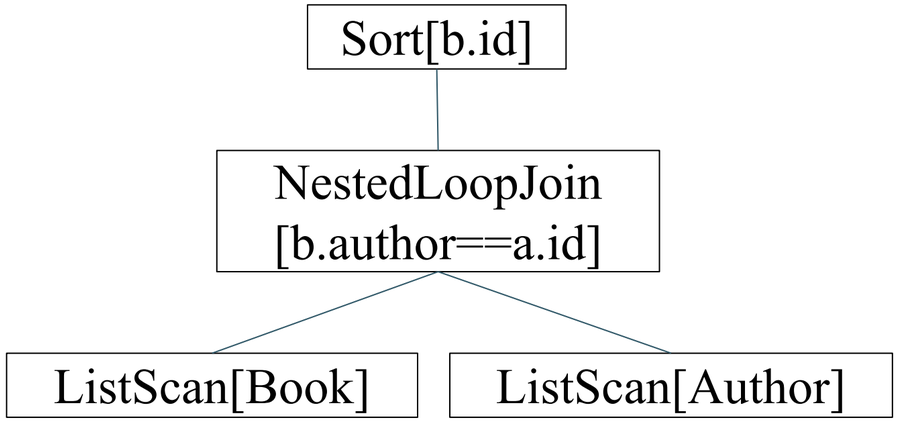
Naive implementation

```
List<Tuple> buildMainView(List<Book> books, List<Author> authors, int limit)
{
   List<Tuple> result = new ArrayList<>();
   for (Book b : books)
   {
      boolean hasAuthor = false;
      for (Author a : authors)
         if (b.author == a.id)
            hasAuthor = result.add(Tuple.of(b, a));
      if (!hasAuthor)
         result.add(b);
   }


}
```



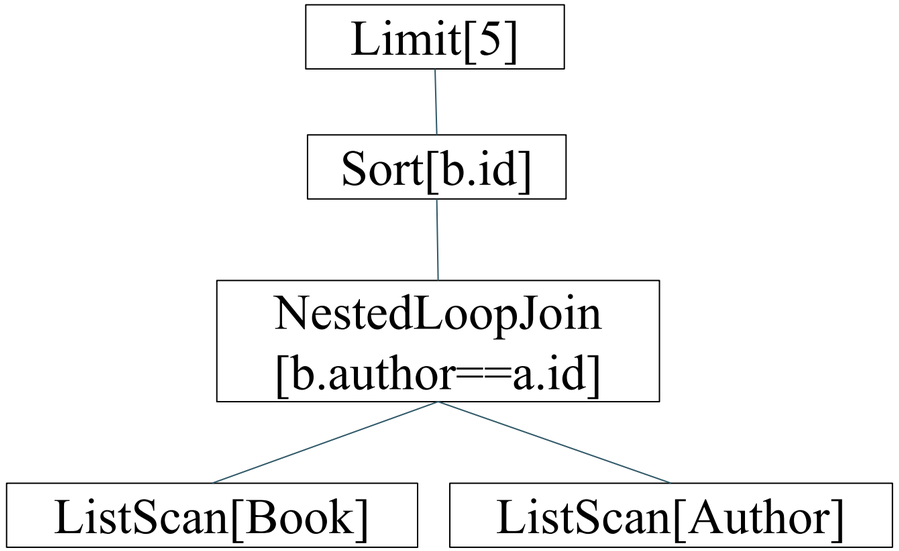
Naive implementation

```
List<Tuple> buildMainView(List<Book> books, List<Author> authors, int limit)
{
   List<Tuple> result = new ArrayList<>();
   for (Book b : books)
   {
      boolean hasAuthor = false;
      for (Author a : authors)
         if (b.author == a.id)
            hasAuthor = result.add(Tuple.of(b, a));
      if (!hasAuthor)
         result.add(b);
   }
   Collections.sort(result, Comparator.comparing(t->t.get("b.id", Integer.class)));

}
```



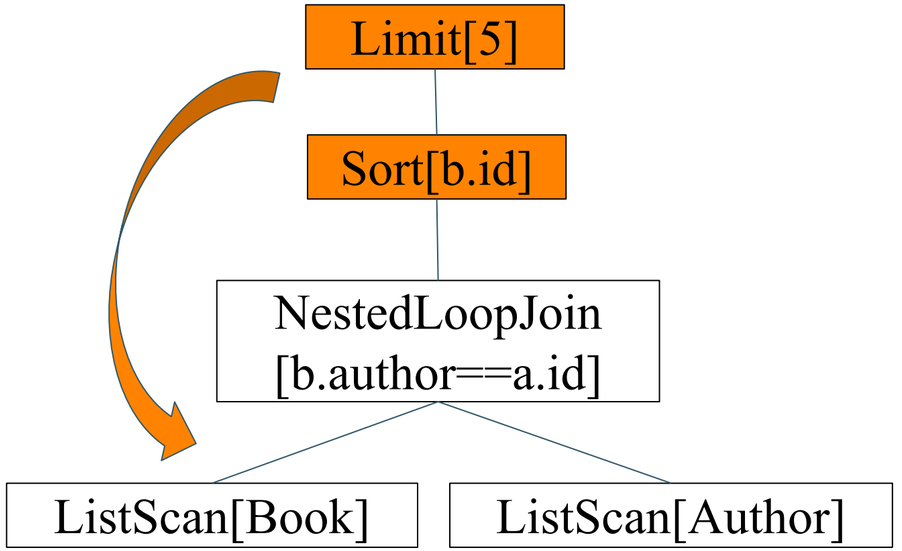
Naive implementation

```
List<Tuple> buildMainView(List<Book> books, List<Author> authors, int limit)
{
   List<Tuple> result = new ArrayList<>();
   for (Book b : books)
   {
      boolean hasAuthor = false;
      for (Author a : authors)
         if (b.author == a.id)
            hasAuthor = result.add(Tuple.of(b, a));
      if (!hasAuthor)
         result.add(b);
   }
   Collections.sort(result, Comparator.comparing(t->t.get("b.id", Integer.class)));
   return result.subList(0, Math.min(limit, result.size()));
}
```



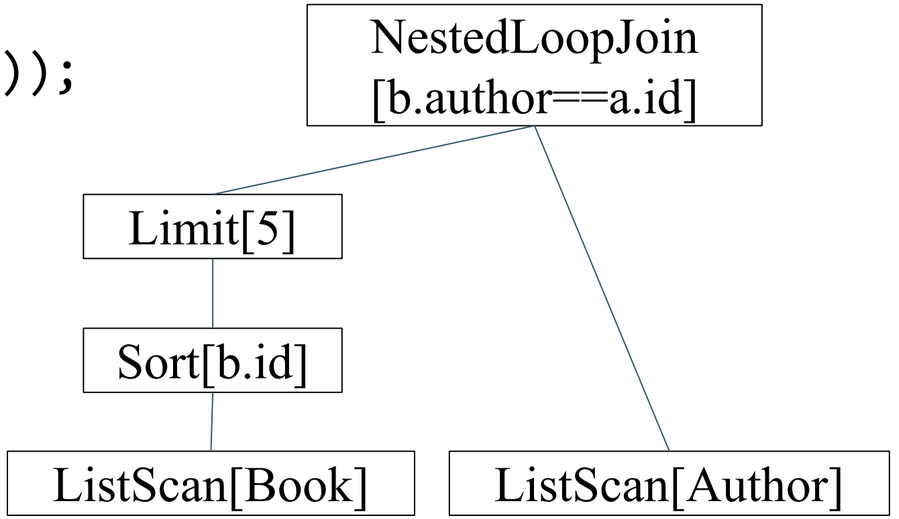
Operations can be re-ordered

```
List<Tuple> buildMainView(List<Book> books, List<Author> authors, int limit)
{
   List<Tuple> result = new ArrayList<>();
   for (Book b : books)
   {
      boolean hasAuthor = false;
      for (Author a : authors)
         if (b.author == a.id)
            hasAuthor = result.add(Tuple.of(b, a));
      if (!hasAuthor)
         result.add(b);
   }
   Collections.sort(result, Comparator.comparing(t->t.get("b.id", Integer.class)));
   return result.subList(0, Math.min(limit, result.size()));
}
```



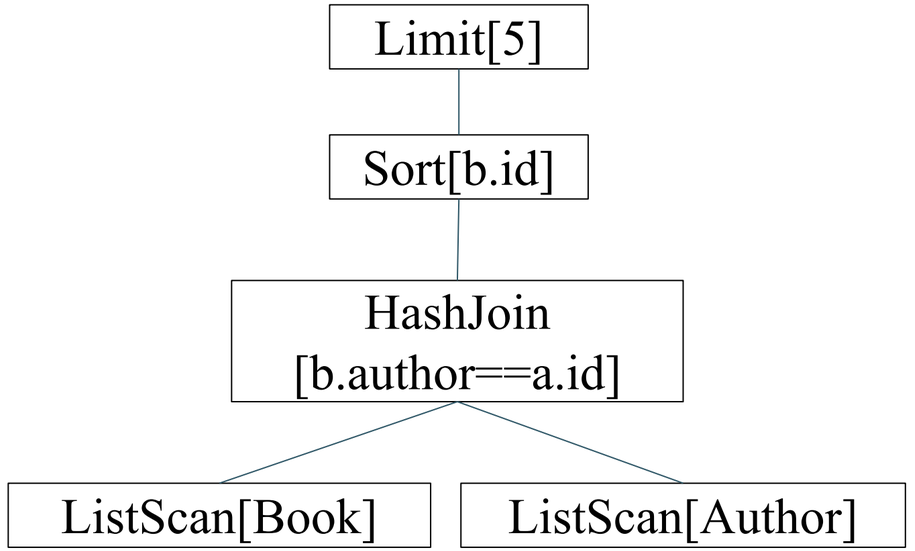
Operations can be re-ordered

```
List<Tuple> buildMainView(List<Book> books, List<Author> authors, int limit)
{
    List<Tuple> result = new ArrayList<>();
   Collections.sort(books, Comparator.comparing(b -> b.id));
   for (Book b : books)
   {
      if (result.size()==limit)
         break;
      boolean hasAuthor = false;
      for (Author a : authors)
         if (b.author == a.id)
            hasAuthor = result.add(Tuple.of(b, a));
      if (!hasAuthor)
         result.add(b);
   }
   return result;
}
```



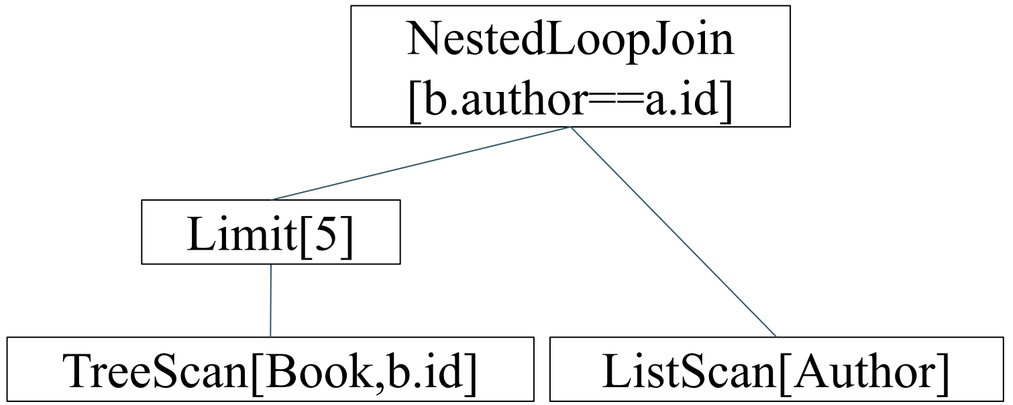
Operations can be implemented differently

```
List<Tuple> buildMainView(List<Book> books, List<Author> authors, int limit)
{
   List<Tuple> result = new ArrayList<>();
    Map<Integer, Author> ixAuthor = new HashMap<>();
   for (Author a : authors)
      ixAuthor.put(a.id, a);
   for (Book b : books)
   {
      Author a = ixAuthor.get(b.author);
      if (a == null)
         result.add(b);
      else
         result.add(Tuple.of(b, a));
   }
   Collections.sort(result, Comparator.comparing(t->t.get("b.id", Integer.class)));
   return result.subList(0, Math.min(limit, result.size()));
}
```



Multiple ways of accessing data

```
List<Tuple> buildMainView(TreeMap<Integer,Book> books, List<Author> authors, int
limit)
{
  List<Tuple> result = new ArrayList<>();
   for (Book b : books.values())
   {
      if (result.size()==limit)
         break;
      boolean hasAuthor = false;
      for (Author a : authors)
         if (b.author == a.id)
            hasAuthor = result.add(Tuple.of(b, a));
      if (!hasAuthor)
         result.add(b);
   }
   return result;
}
```



Observations

Observations

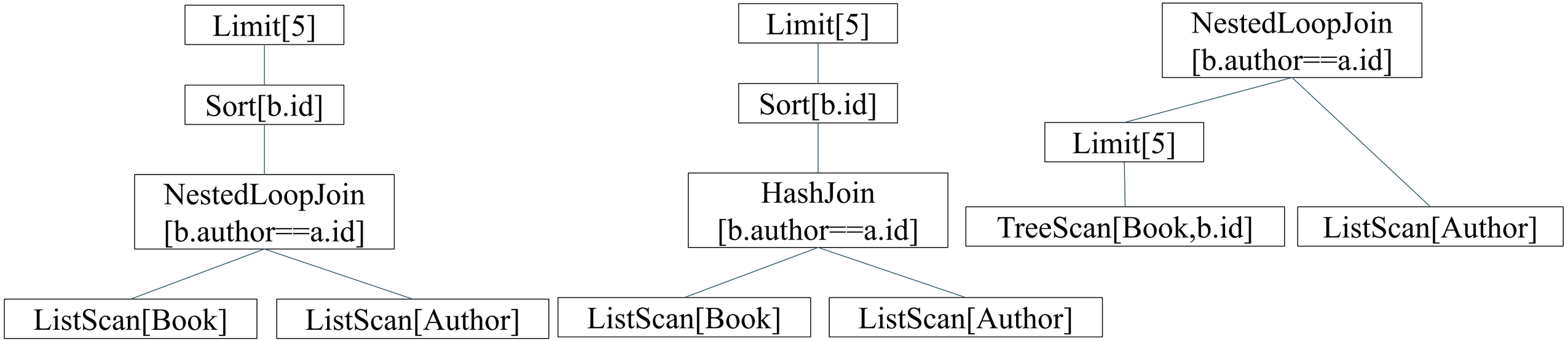
1. Declarative query language

```
SELECT b.id, b.title, b.year, a.fname, a.lname
FROM Book b
LEFT OUTER JOIN Author a ON b.author=a.id
ORDER BY b.id
LIMIT 5
```

Observations

1. Declarative query language
2. Algebraic operators

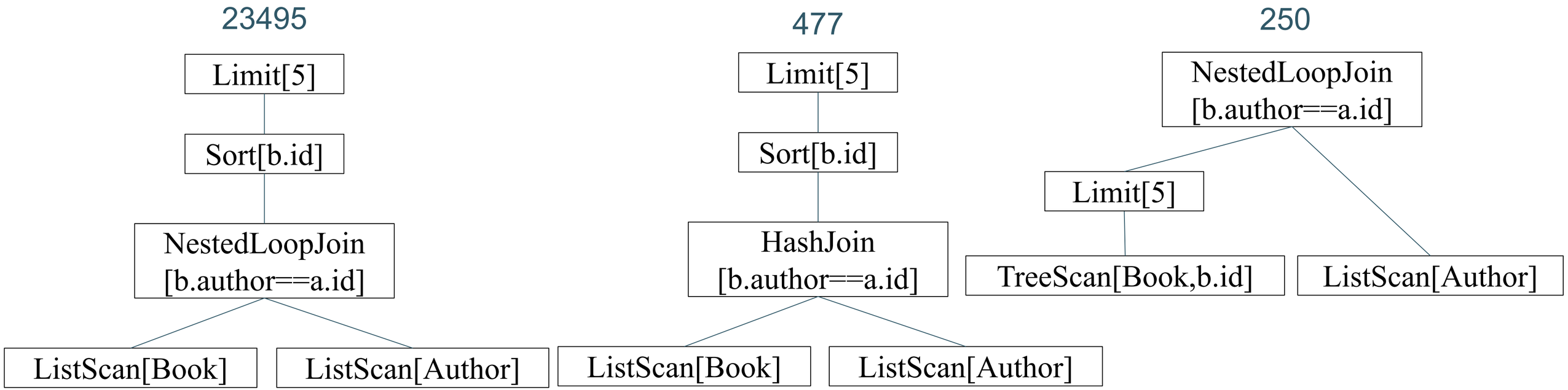
```
SELECT b.id, b.title, b.year, a.fname, a.lname
FROM Book b
LEFT OUTER JOIN Author a ON b.author=a.id
ORDER BY b.id
LIMIT 5
```



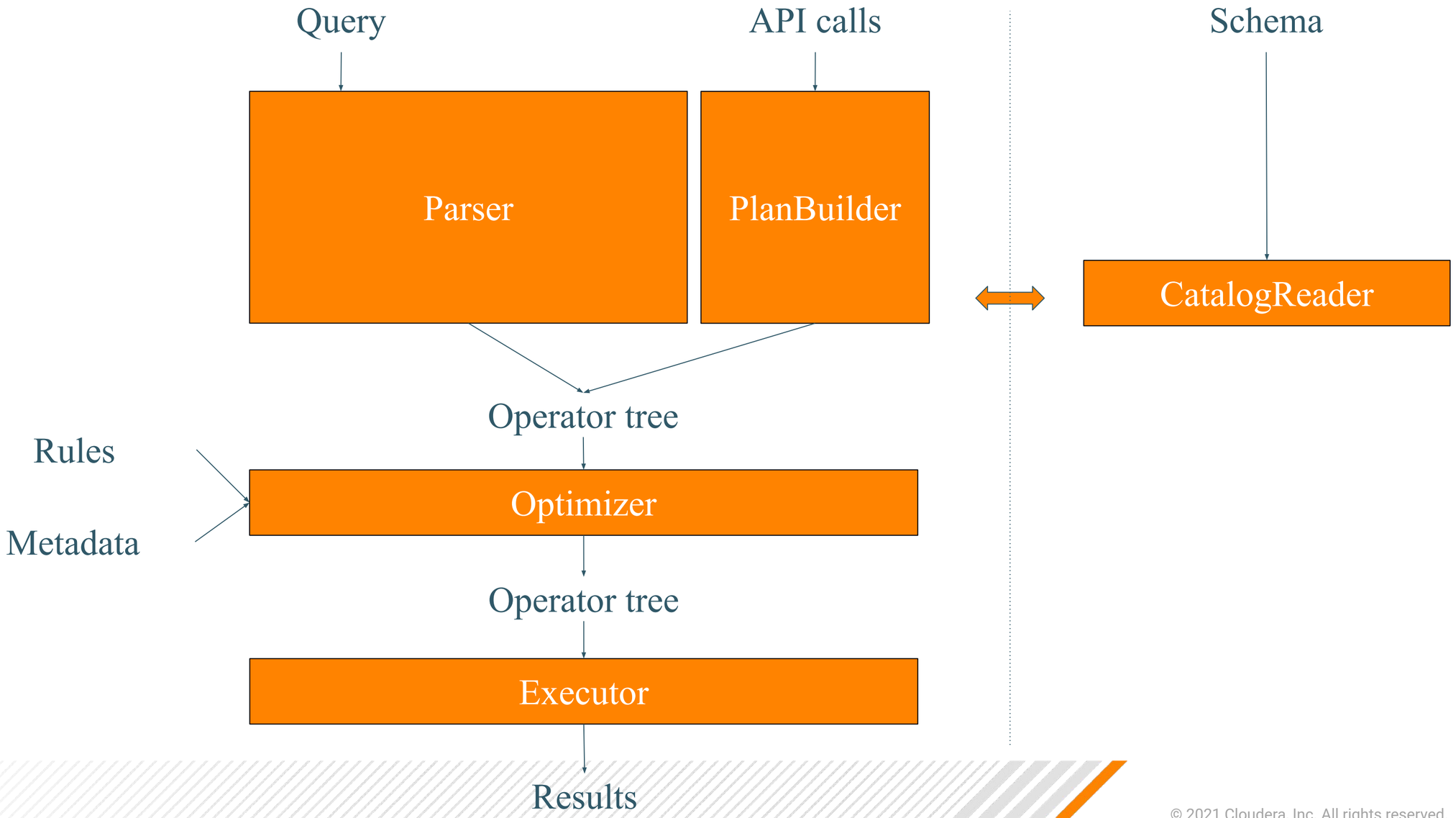
Observations

1. Declarative query language
2. Algebraic operators
3. Query optimizer

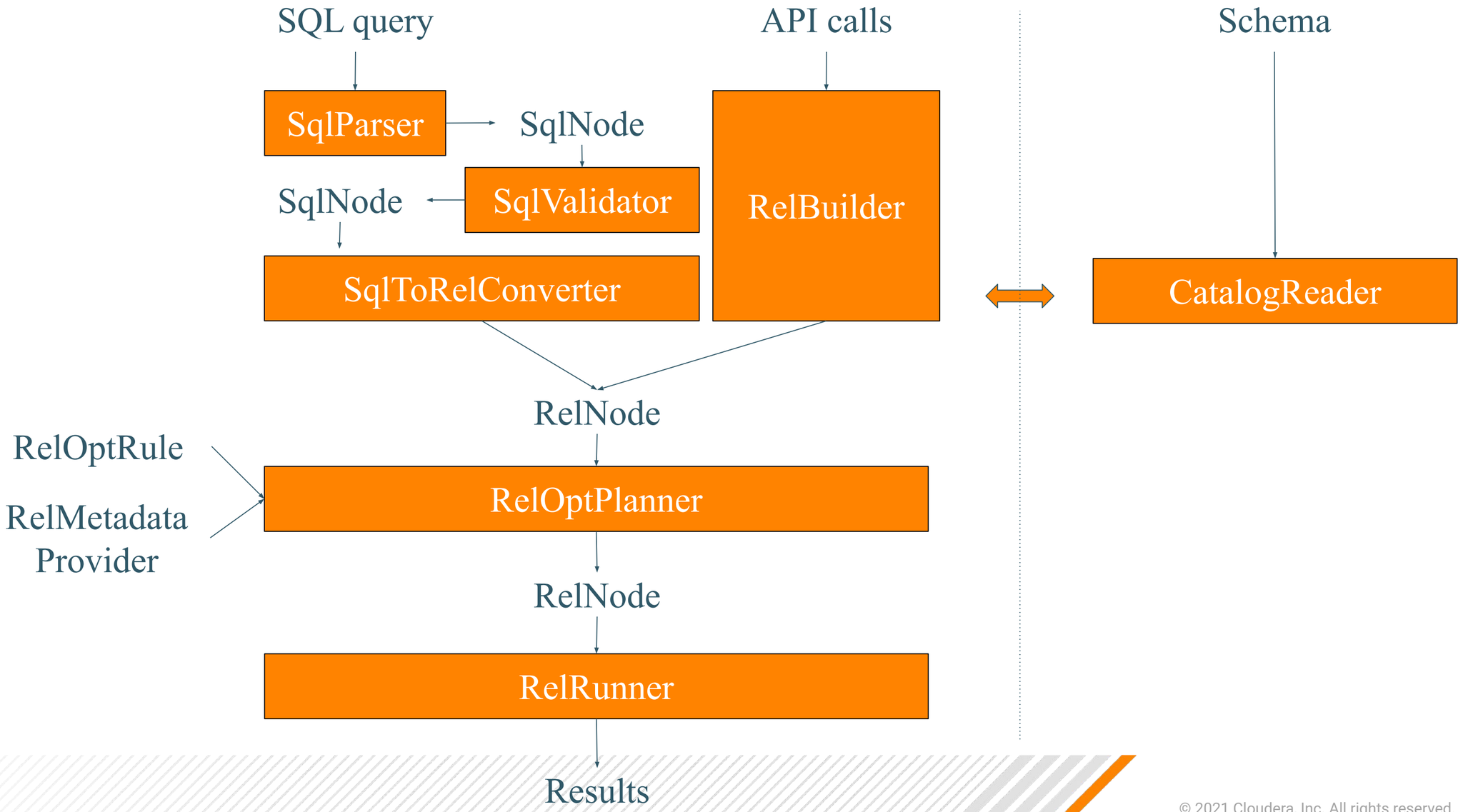
```
SELECT b.id, b.title, b.year, a.fname, a.lname
FROM Book b
LEFT OUTER JOIN Author a ON b.author=a.id
ORDER BY b.id
LIMIT 5
```



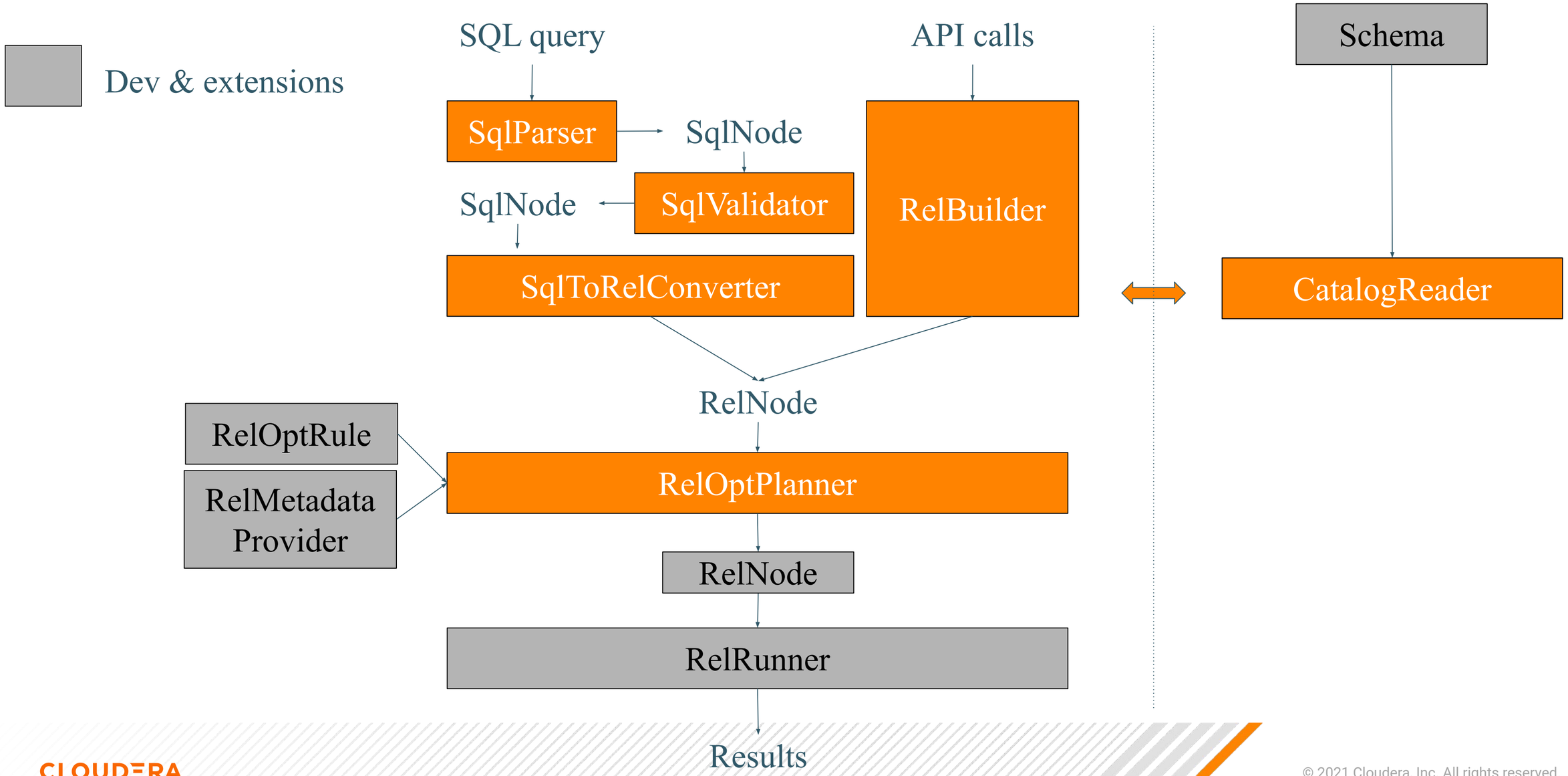
Query processor architecture



Apache Calcite



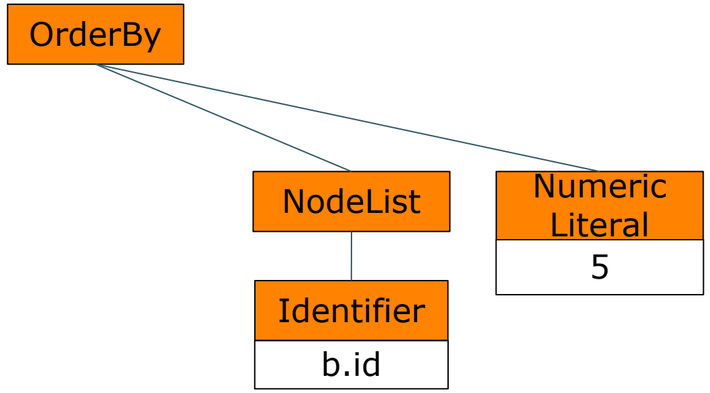
Apache Calcite



Example: Query to AST


```
SELECT b.id, b.title, b.year, a.fname, a.lname
FROM Book b
LEFT OUTER JOIN Author a ON b.author=a.id
WHERE b.year > 1830
ORDER BY b.id
LIMIT 5
```

Example: Query to AST

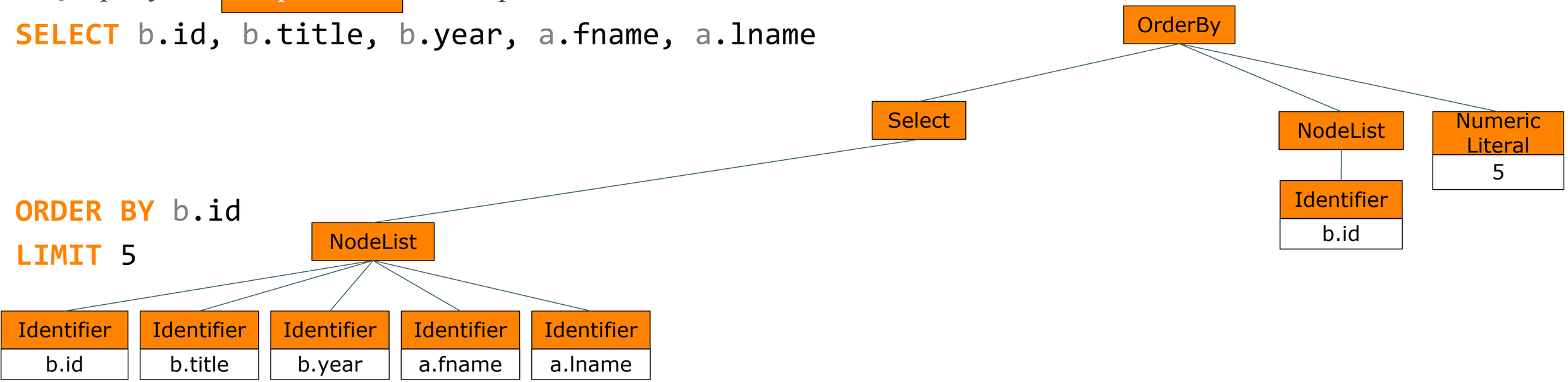



```
ORDER BY b.id
LIMIT 5
```

Example: Query to AST

SQL query → SqlParser → SqlNode

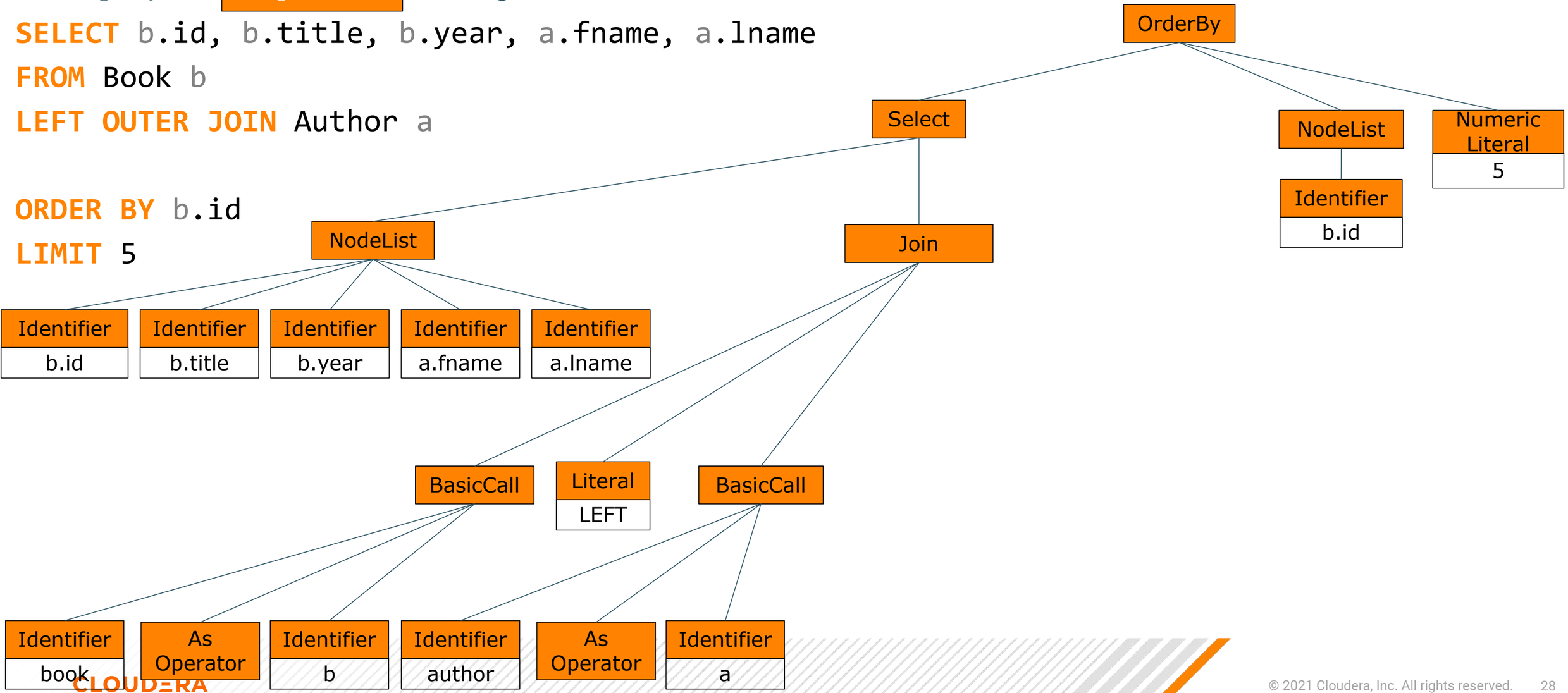
```
SELECT b.id, b.title, b.year, a.fname, a.lname
ORDER BY b.id
LIMIT 5
```



Example: Query to AST

SQL query → SqlParser → SqlNode

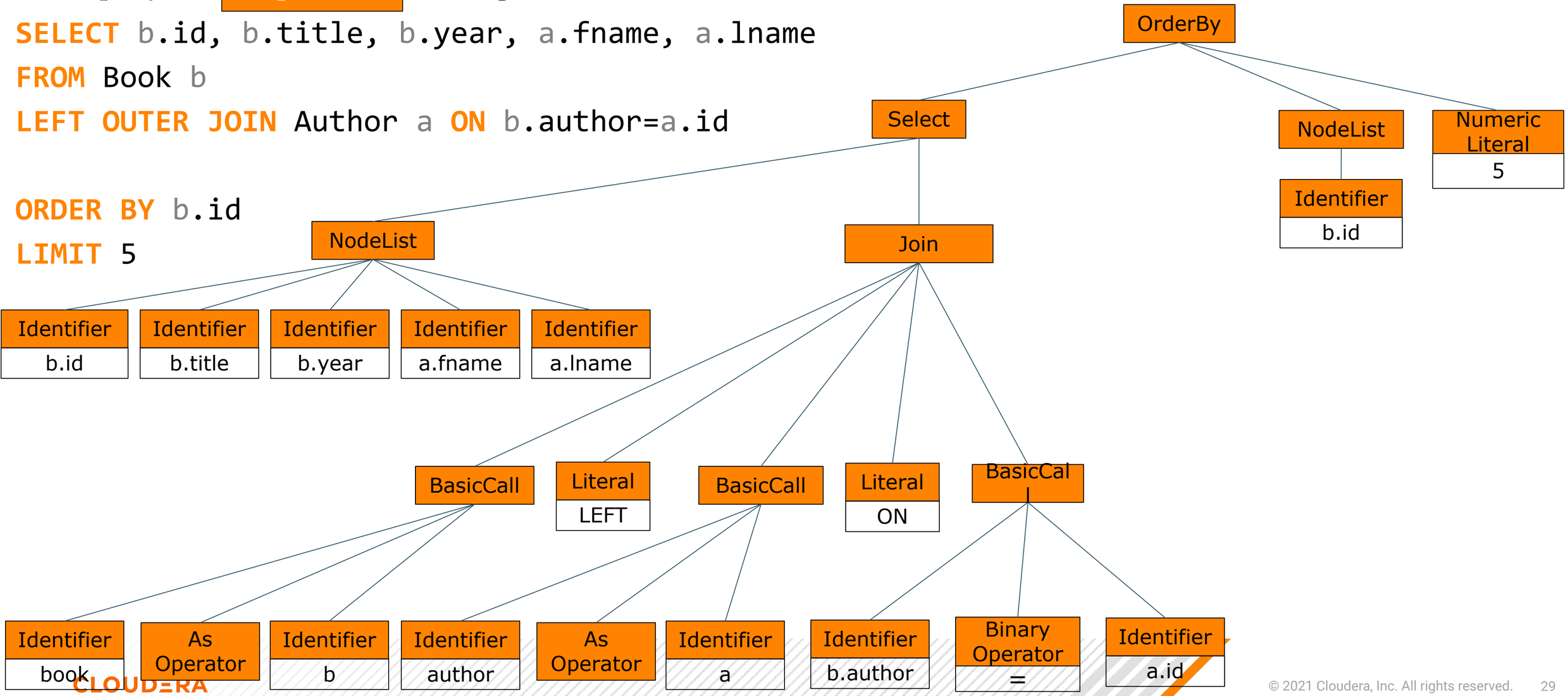
```
SELECT b.id, b.title, b.year, a.fname, a.lname
FROM Book b
LEFT OUTER JOIN Author a

ORDER BY b.id
LIMIT 5
```



Example: Query to AST

SQL query → SqlParser → SqlNode

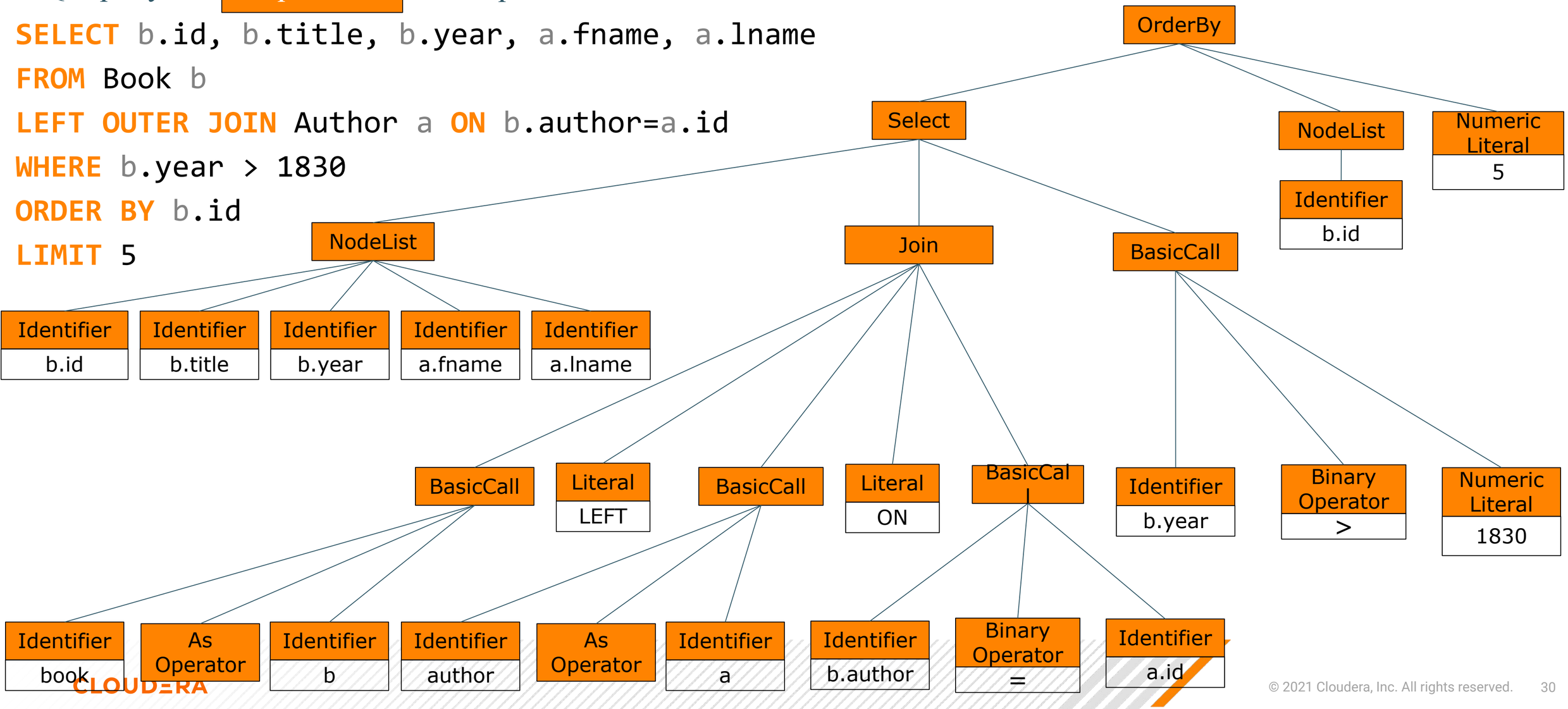
```
SELECT b.id, b.title, b.year, a.fname, a.lname
FROM Book b
LEFT OUTER JOIN Author a ON b.author=a.id

ORDER BY b.id
LIMIT 5
```



Example: Query to AST

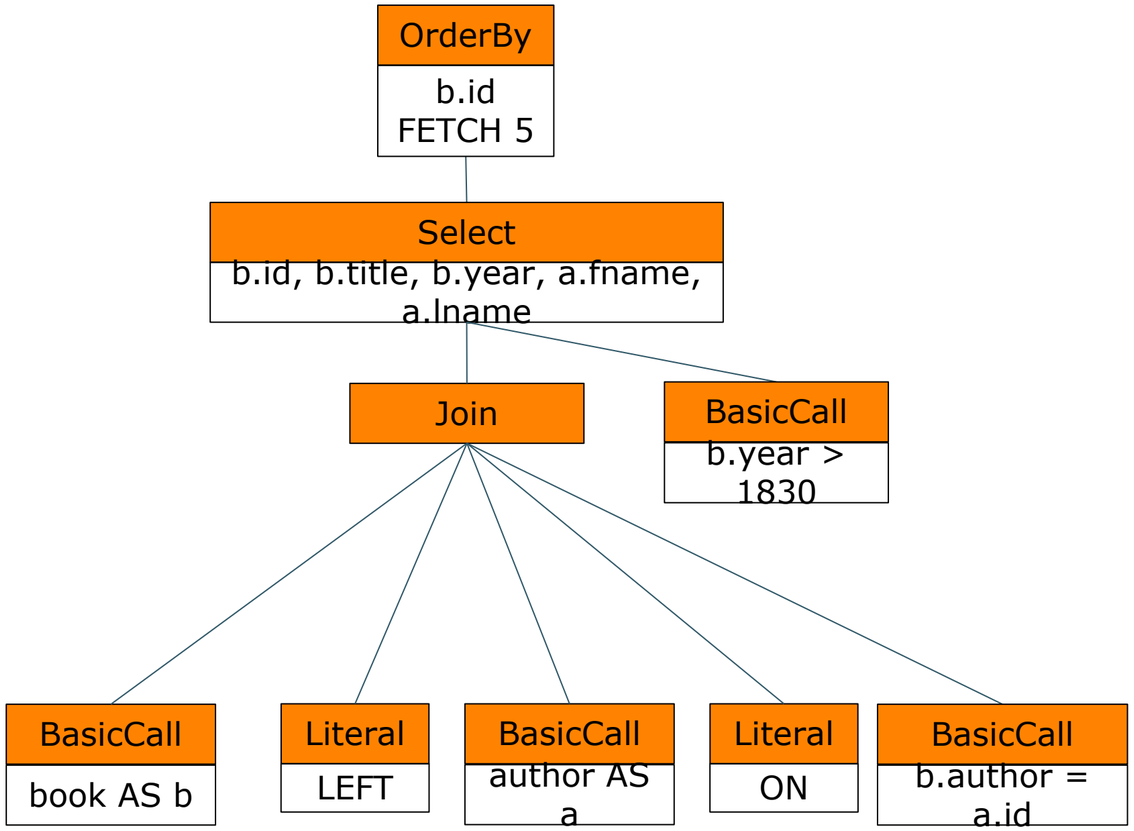
SQL query → SqlParser → SqlNode

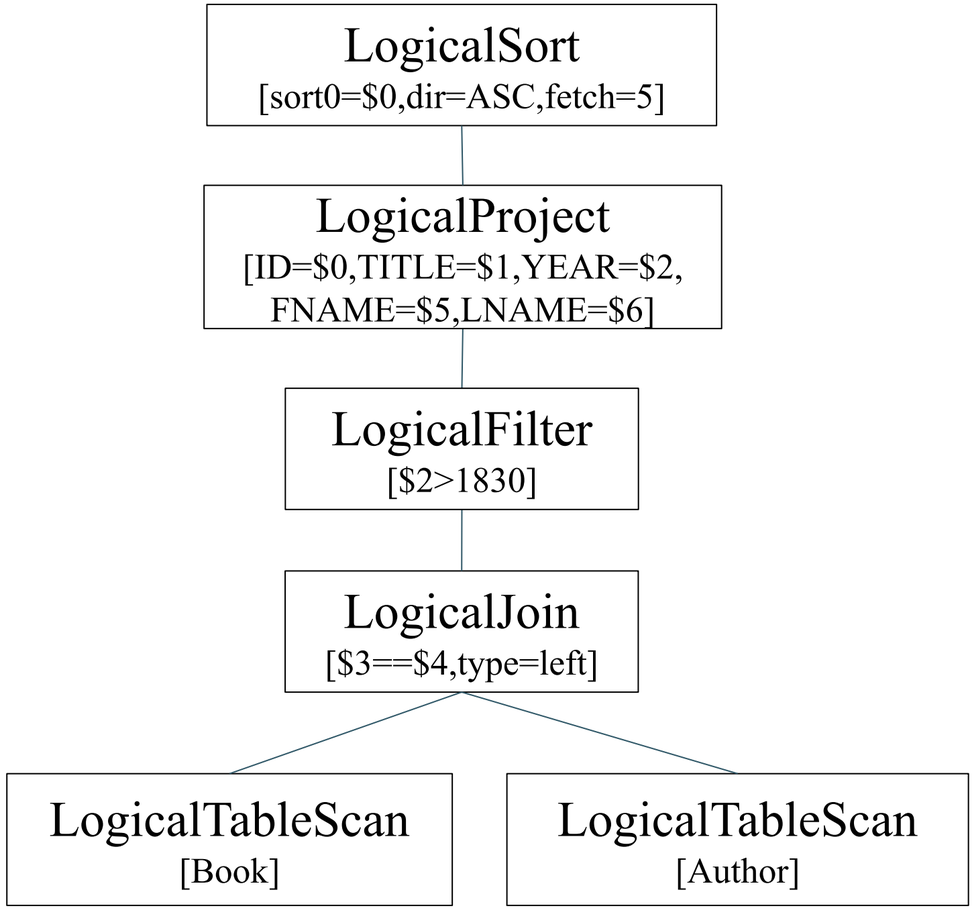
```
SELECT b.id, b.title, b.year, a.fname, a.lname
FROM Book b
LEFT OUTER JOIN Author a ON b.author=a.id
WHERE b.year > 1830
ORDER BY b.id
LIMIT 5
```



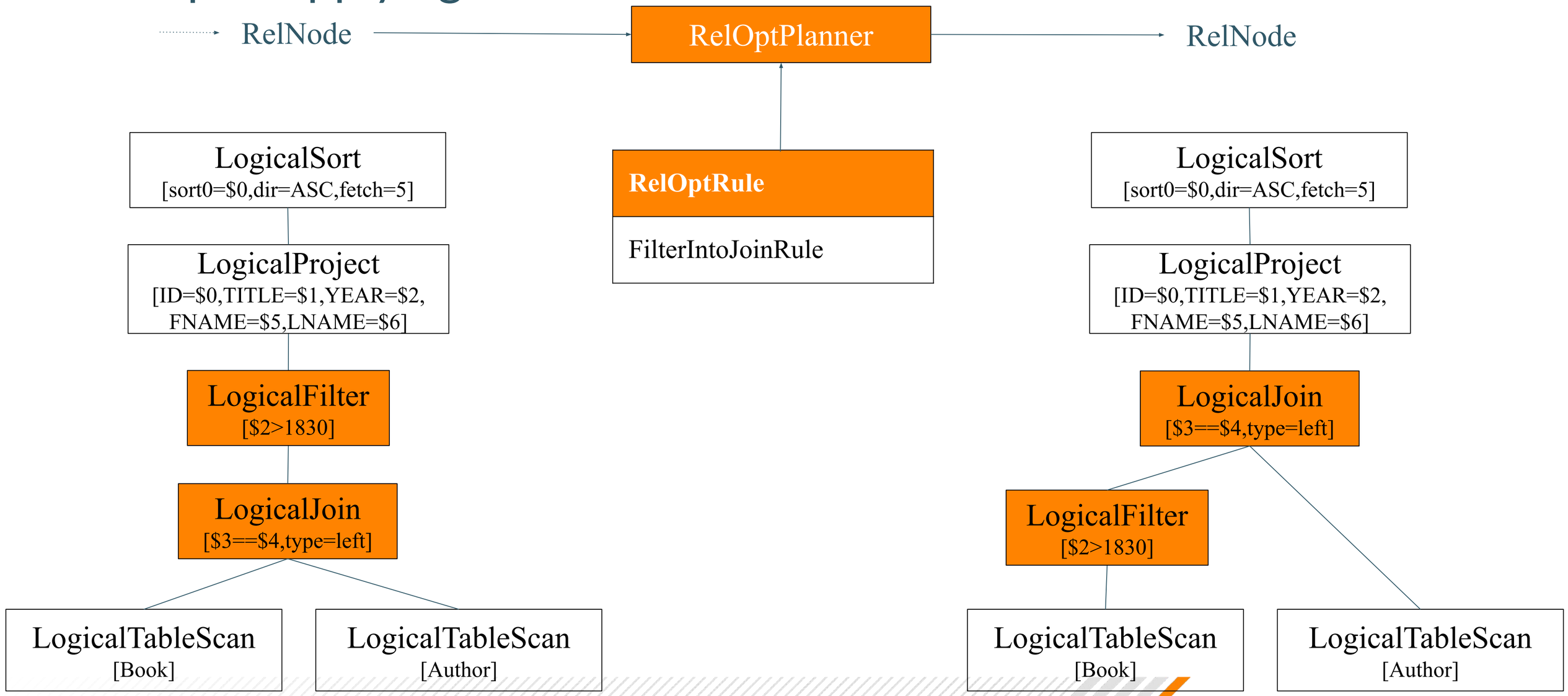
Example: AST to logical plan

SqlNode → **SqlValidator** → SqlNode → **SqlToRelConverter** → RelNode

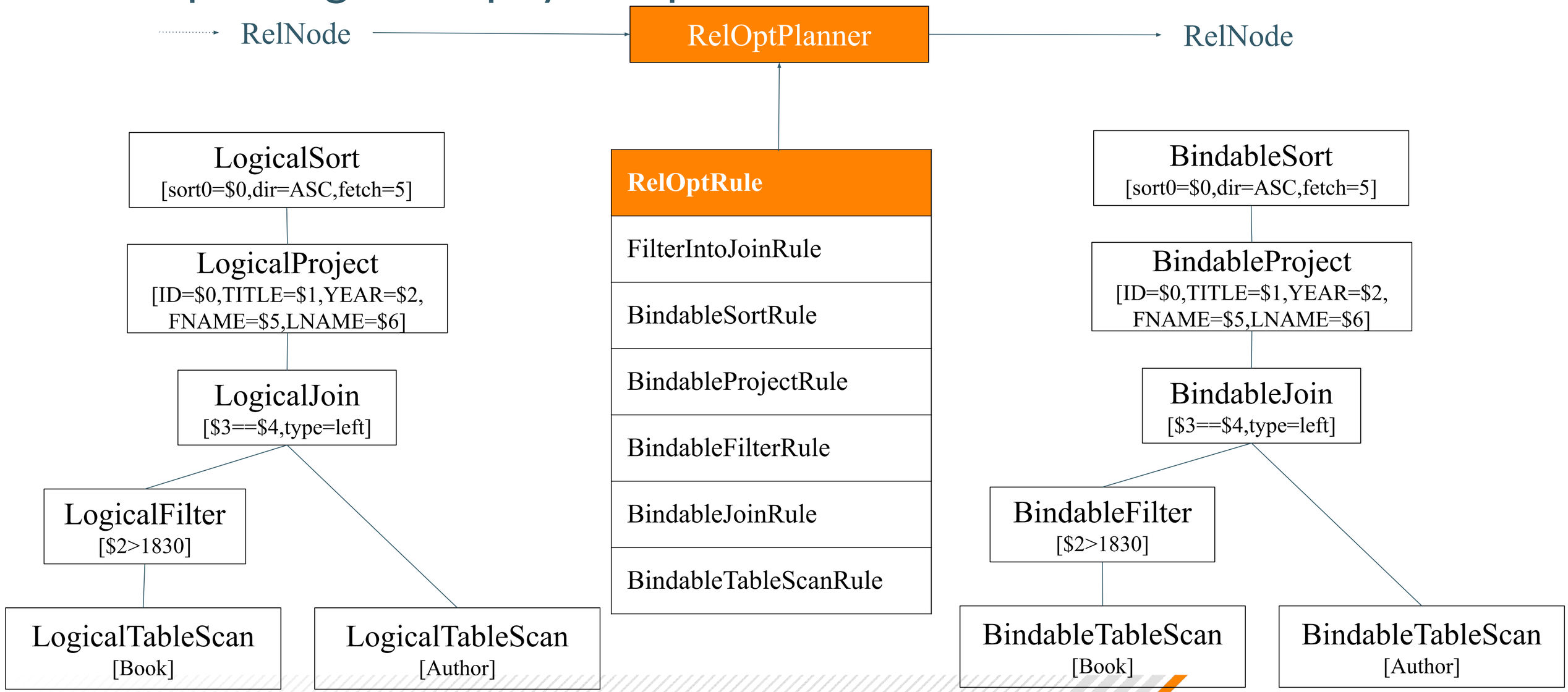




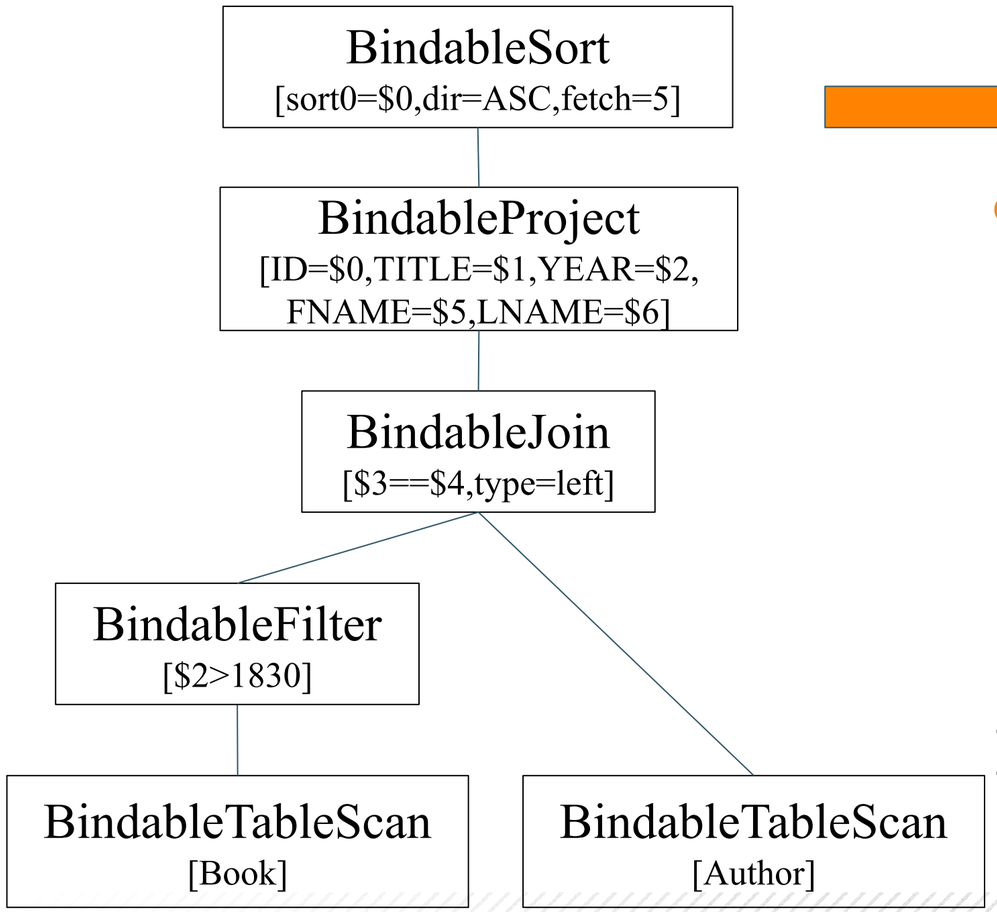
Example: Applying rules



Example: Logical to physical plan



Example: Physical plan execution




```
class BindableExecutor {
     List<Object[]> execute(BindableRel node,
                            DataContext context)
     {
        List<Object[]> result = new ArrayList<>();
       for (Object[] tuple:node.bind(context))
         result.add(tuple);
        return result;
     }
}
```

Powered by Calcite