Beyond Hive: Standard SQL on Hadoop

Software and Services Group

Agenda

Introduce of "Project Panthera ASE"

Design and Architecture

Performance

Test Suite

Summary

SQL in Hadoop Eco-system

Hive

Impala

Presto

Phoenix

Shark

…

Full Analytic SQL Support for Hadoop Needed

Full SQL support for OLAP

- Required in modern business application environment
  - Business users
  - Enterprise analytics applications
  - Third-party tools (such as query builders and BI applications)

Hive – *THE* Data Warehouse for Hadoop

- HiveQL: a SQL-like query language (subset of SQL with extensions)
  - Significantly lowers the barrier to MapReduce
- Still large gaps with full analytic SQL support
  - Multiple-table SELECT statement, subquery in WHERE and HAVING clauses, etc.
  - INTERSECT, MINUS, UNION, Natural Join, etc.

Project Panthera ASE

Our open source efforts to enable better *analytics capabilities* on Hadoop

- Built on top of Hive
- Provide full SQL support for OLAP
- Better integration with existing infrastructure using SQL
- Etc.

Agenda

Introduce of "Project Panthera ASE"

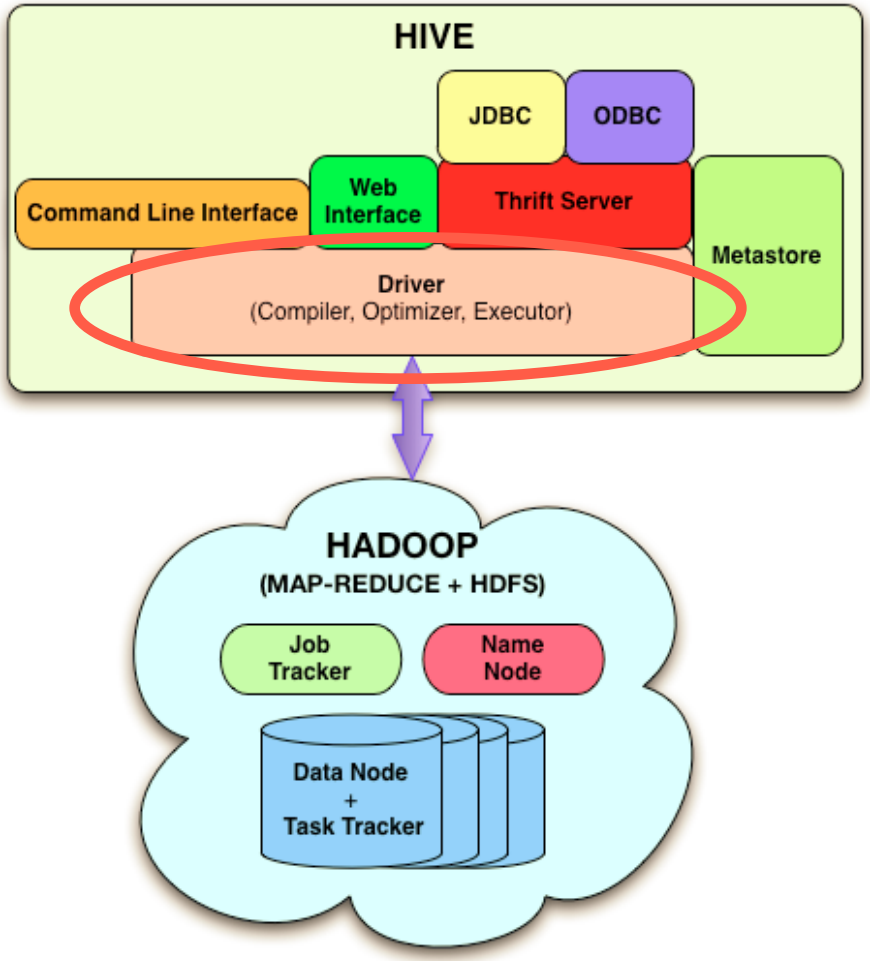
Design and Architecture

Performance

Test Suite

Summary

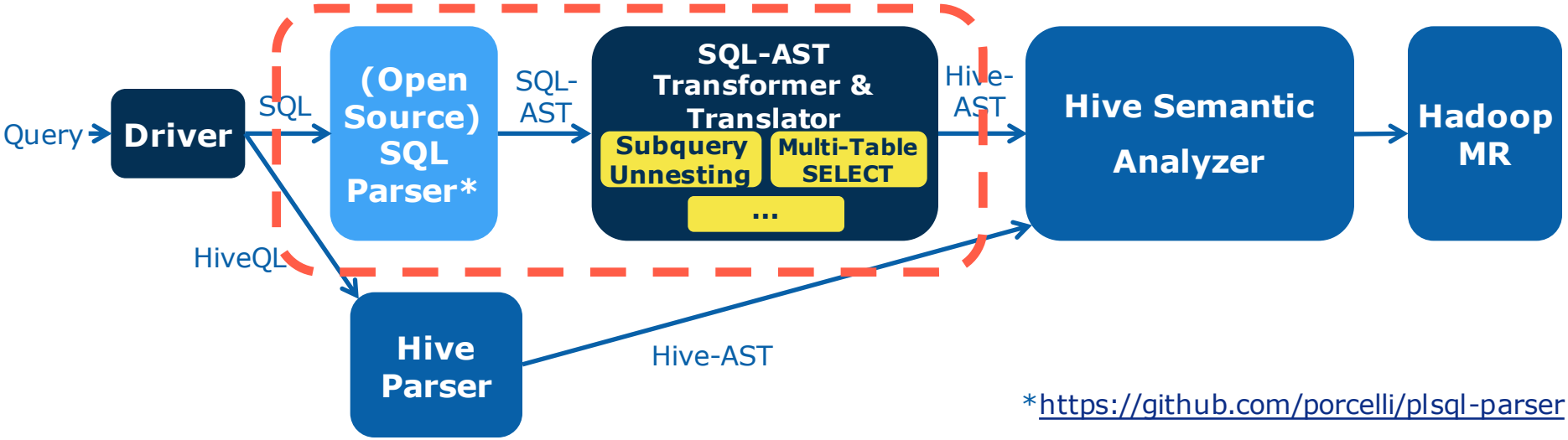
Hive Architecture



An analytical SQL engine for MapReduce

The anatomy of a query processing engine

Our SQL engine for MapReduce



*https://github.com/porcelli/plsql-parser

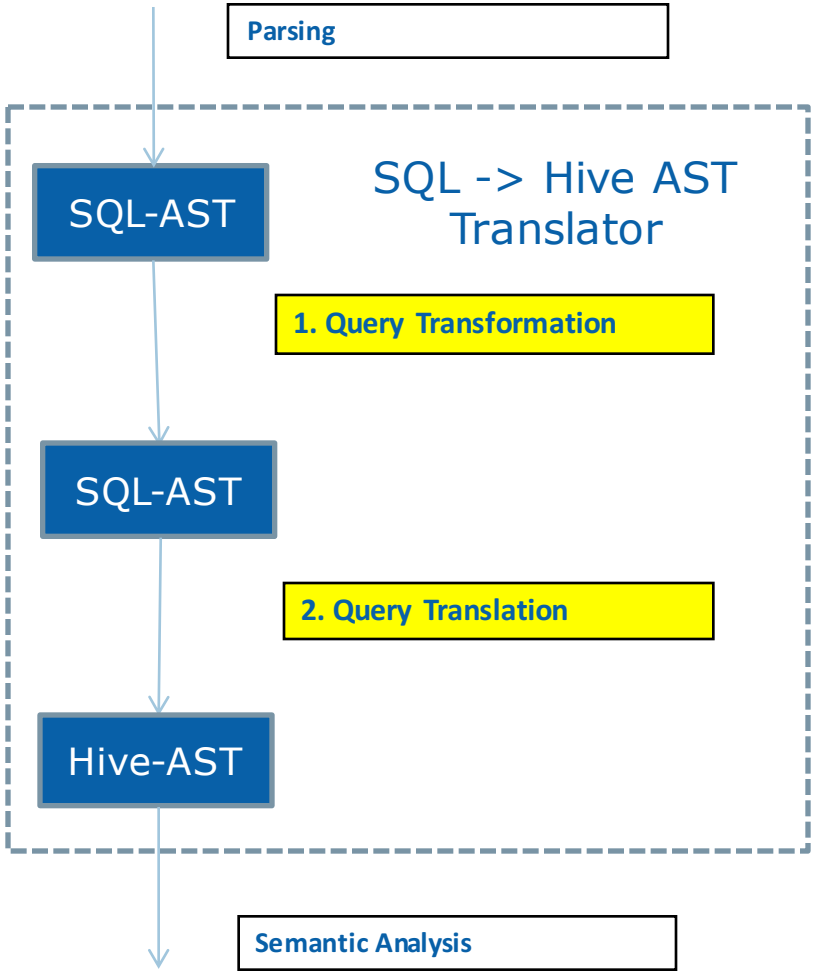
Overall Architecture

1. Query Transformation

- Syntax Transformations
  - Multiple-table Selection Transformation
  - Subquery Unnesting
    - Build Filter Block Tree
    - AST Transformation
  - Other Transformations
- Alias Fixing

2. Query Translation Stage

- Do nearly node-to-node mapping from SQL-AST to Hive-AST



Multiple Table Selection

Example Query

- SELECT * FROM t1, t2, t3;
- SELECT COUNT(*) FROM STAFF1,WORKS1,PROJ1 WHERE STAFF1.EMPNUM = 'E9' AND STAFF1.EMPNUM = WORKS1.EMPNUM AND PROJ1.PNUM = WORKS1.PNUM;

***Solution*: In general , translate multi-table selection to cross JOIN. If condition is equal expressions, optimized it to inner join e.g.**

- SELECT * FROM t1 CROSS JOIN t2 CROSS JOIN t3;
- SELECT COUNT(*) FROM STAFF1 JOIN WORKS1 ON STAFF1.EMPNUM = WORKS1.EMPNUM JOIN PROJ1 ON PROJ1.PNUM = WORKS1.PNUM WHERE STAFF1.EMPNUM = 'E9';

SubQ Unnesting - Filter Definition

Filters are SQL conditions used in WHERE/HAVING clauses. Each filter returns TRUE or FALSE on an input row and it determines whether or not the input row is qualified for further processing. Filter is defined as below:

- ***Filter*** **:=** ***SimpleFilter*** **|** ***Filter*** **AND** ***Filter*** **|** ***Filter*** **OR** ***Filter*** **|** **NOT** ***Filter***
- ***SimpleFilter*** *:= SimpleNormalFilter | SimpleSubQueryFilter*
- ***SimpleNormalFilter*** :=
  - *SimpleExpression* [>|<|>=|<=|!=|=] *SimpleExpression*
  - | *SimpleExpression IS NULL* | *SimpleExpression IS NOT NULL*
  - | *SimpleExpression IN (SimpleExpression List)* | *SimpleExpression NOT IN (SimpleExpression List)*
  - | All other expressions that returns Boolean (e.g. *Pattern matching condition (LIKE…ESCAPE clause), range condition (BETWEEN…AND clause), case condition (CASE…WHEN clause), etc.)*
  - *SimpleNormalFilter which refers to column that does not belongs to the direct enclosing query, is called* ***CorrelatedSimpleNormalFilter****, otherwise it is called* ***UncorrelatedSimpleNormalFilter***
- ***SimpleSubQueryFilter*** :=
  - *SimpleExpression* [>|<|>=|<=|!=|=] *SubQuery*
  - | *SubQuery* [>|<|>=|<=|!=|=] *SimpleExpression*
  - | *ISNULL (SubQ)* | *ISNOTNULL (SimpleExpression)*
  - | *SimpleExpression IN (SubQuery)* | *SimpleExpression NOT IN (SubQuery)*
  - | *SimpleExpression* [>|<|>=|<=|!=|=] [ALL|SOME|ANY] *(SubQuery)*
  - | *Exists (SubQuery)*
- ***SimpleExpression*** *:=*
  - *Literal (e.g. 1,2, "abc")*
  - |*ColumnRef (e.g a, t1.b,db.t2.c)*
  - | *UDF/UDAF of SimpleExpression (e.g. substr(a,1,2), sin(), regex_replace(substr(…)), etc.)*

SubQ Unnesting - Filter Definition

- *UncorrelatedNormalFilter* :=
  *Uncorrelated**Simple**NormalFilter*
  | NOT *UncorrelatedNormalFilter*
  | *UncorrelatedNormalFilter* AND *UncorrelatedNormalFilter*
  | *UncorrelatedNormalFilter* OR *UncorrelatedNormalFilter*
- *CorrelatedNormalFilter* :=
  *Correlated**Simple**NormalFilter*
  | NOT *CorrelatedNormalFilter*
  | *CorrelatedNormalFilter* AND *CorrelatedNormalFilter*
  | *CorrelatedNormalFilter* OR *CorrelatedNormalFilter*

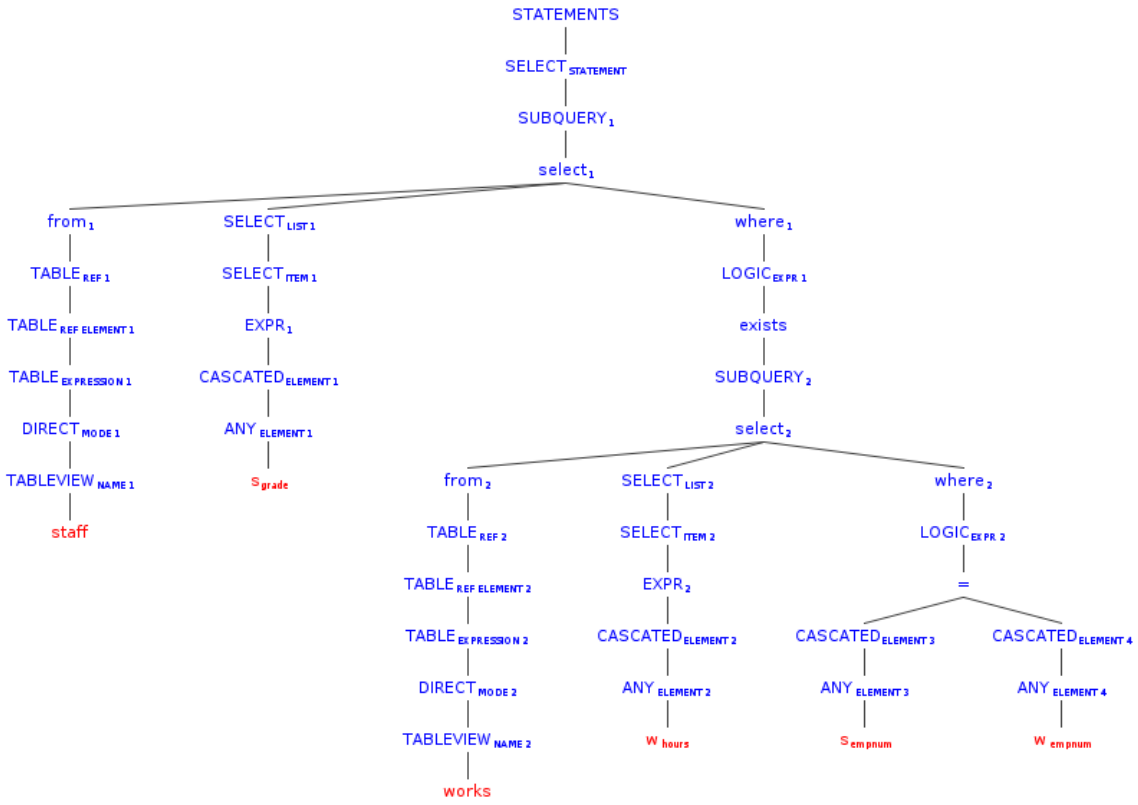
SubQ Unnesting Specification

EXISTS:

- W/ Correlated Filter: Select a from t1 where exists (select b from t2 where t2.c = t1.c) → select a from t1 left semi join (select b as col0, c as col1 from t2) subq1 on subq1.col1=t1.c
- W/O Correlated Filter: Select a from t1 where exists (select b from t2 where t2.c > 100) → select a from t1 left semi join (select b from t2 where t2.c > 100) subq1

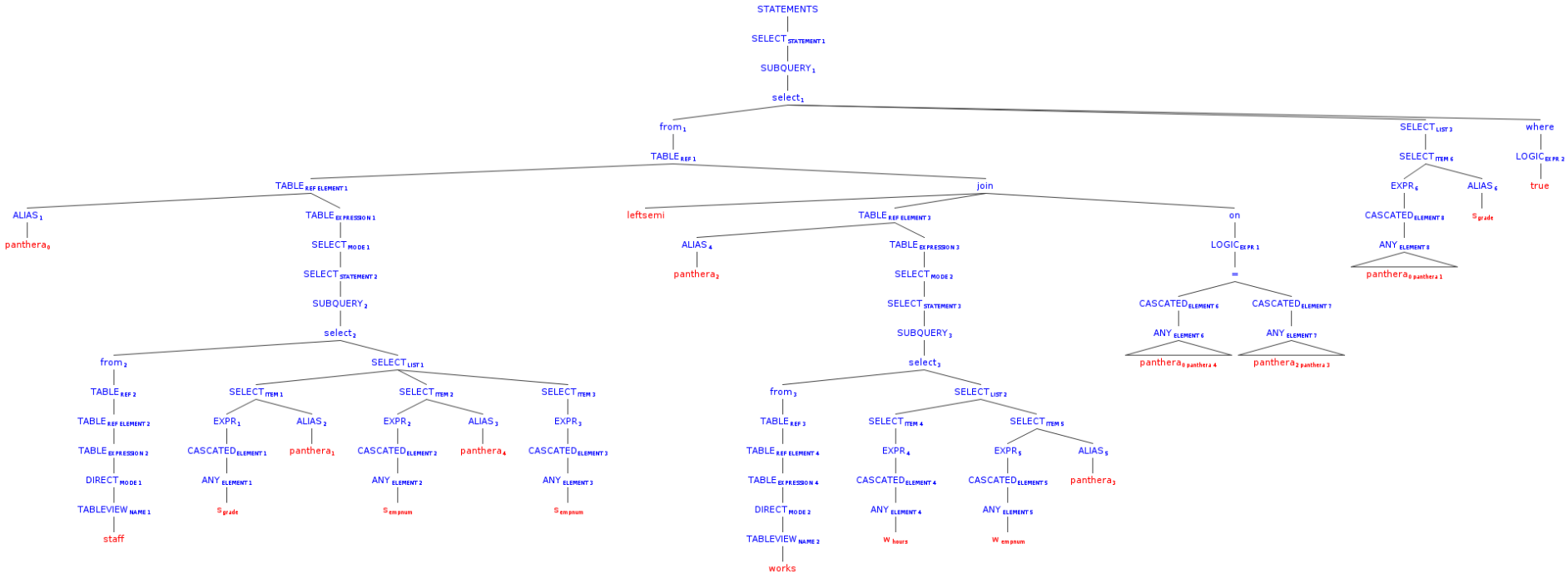
Exists Case Study

select s_grade from staff where exists (select w_hours from works where s_empnum=w_empnum);



Exists Case Study

select panthera_0.panthera_1 as s_grade from (select s_grade as panthera_1, s_empnum as panthera_4, s_empnum from staff) panthera_0 left semi join (select w_hours, w_empnum as panthera_3 from works) panthera_2 on panthera_0.panthera_4 = panthera_2.panthera_3 where true ;



SubQ Unnesting Specification

NOT EXISTS:

- W/ Correlated Filter: Select a from t1 where not exists (select b from t2 where t2.c = t1.c) → select a from ((select t1._rowid, a from t1) subq2 MINUS (select distinct t1._rowid, a from t1 join (select b as col1, c as col2 from t2) subq1 on subq1.col2 = t1.c) subq3) subq4
- W/O Correlated Filter: Select a from t1 where not exists (select b from t2 where t2.c > 100) → select a from t1 join (select count(*) as count from (select b from t2 where t2.c > 100) subq1) subq2 on subq2=0;

SubQ Unnesting Specification

IN:

- W/ Correlated Filter: Select a from t1 where b in (select b from t2 where t2.c = t1.c) → select subq2.col1 from (select t1._rowid as col0, t1.a as col1, t1.b as col2 from t1 left semi join (select b as col1, c as col2 from t2) subq1 where subq1.col2 = t1.c) subq2
- W/O Correlated Filter: Select a from t1 where b in (select b from t2 where t2.c > 100) → select a from t1 left semi join (select b as col0 from t2 where t2.c > 100) subq1 on subq1.col0=t1.b

SubQ Unnesting Specification

NOT IN:

- W/ Correlated Filter: Select a from t1 where b not in (select b from t2 where t2.c = t1.c) → select a from (select t1._rowid, a from t1 MINUS (select t1_rowid, a from t1 where b in (select b from t2 where t2.c = t1.c)subq1)subq2)subq3
- W/O Correlated Filter: Select a from t1 where b not in (select b from t2 where t2.c > 100) → select a from t1 from t1 cross join (select collect_set(b) as b1 from t2 where t2.c > 100) subq1 where not array_contains(subq1.b1, t1.b);

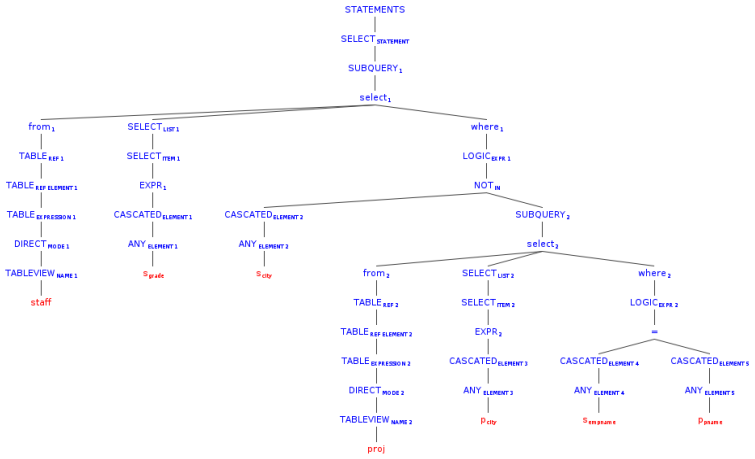
SubQ Unnesting Case Study

select s_grade from staff where s_city not in (select p_city from proj where s_empname=p_pname) →

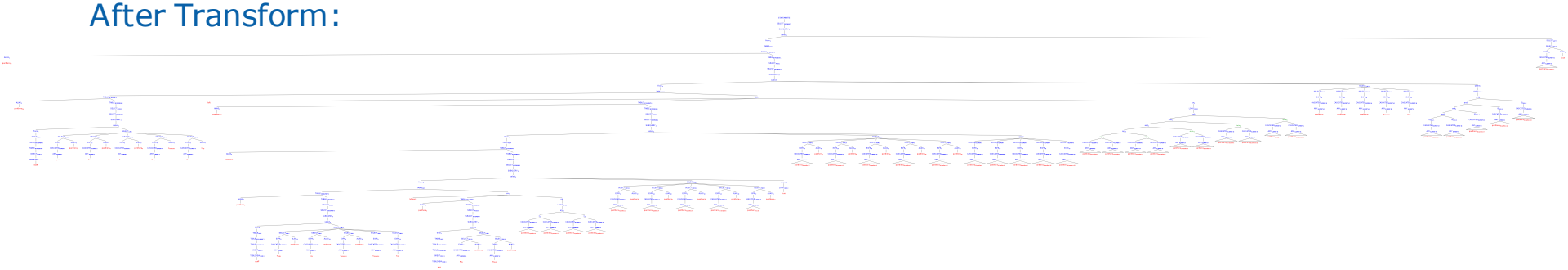
```
select panthera_10.panthera_1 as s_grade from (select panthera_1, panthera_4, panthera_6, s_empname,
s_city from (select s_grade as panthera_1, s_city as panthera_4, s_empname as panthera_6, s_empname as
s_empname, s_city as s_city from staff) panthera_14 left outer join (select panthera_16.panthera_7 as
panthera_7, panthera_16.panthera_8 as panthera_8, panthera_16.panthera_9 as panthera_9,
panthera_16.panthera_12 as panthera_12, panthera_16.panthera_13 as panthera_13 from (select
panthera_0.panthera_1 as panthera_7, panthera_0.panthera_4 as panthera_8, panthera_0.panthera_6 as
panthera_9, panthera_0.s_empname as panthera_12, panthera_0.s_city as panthera_13 from (select s_grade
as panthera_1, s_city as panthera_4, s_empname as panthera_6, s_empname, s_city from staff) panthera_0
left semi join (select p_city as panthera_3, p_pname as panthera_5 from proj) panthera_2 on
(panthera_0.panthera_4 = panthera_2.panthera_3) and (panthera_0.panthera_6 = panthera_2.panthera_5)
where true) panthera_16 group by panthera_16.panthera_7, panthera_16.panthera_8,
panthera_16.panthera_9, panthera_16.panthera_12, panthera_16.panthera_13) panthera_15 on
((((panthera_14.panthera_1 <=> panthera_15.panthera_7) and (panthera_14.panthera_4 <=>
panthera_15.panthera_8)) and (panthera_14.panthera_6 <=> panthera_15.panthera_9)) and
(panthera_14.s_empname <=> panthera_15.panthera_12)) and (panthera_14.s_city <=>
panthera_15.panthera_13) where ((((panthera_15.panthera_7 is null) and (panthera_15.panthera_8 is null))
and (panthera_15.panthera_9 is null)) and (panthera_15.panthera_12 is null)) and
(panthera_15.panthera_13 is null)) panthera_10 ;
```

SubQ Unnesting Case Study

Before Transform:

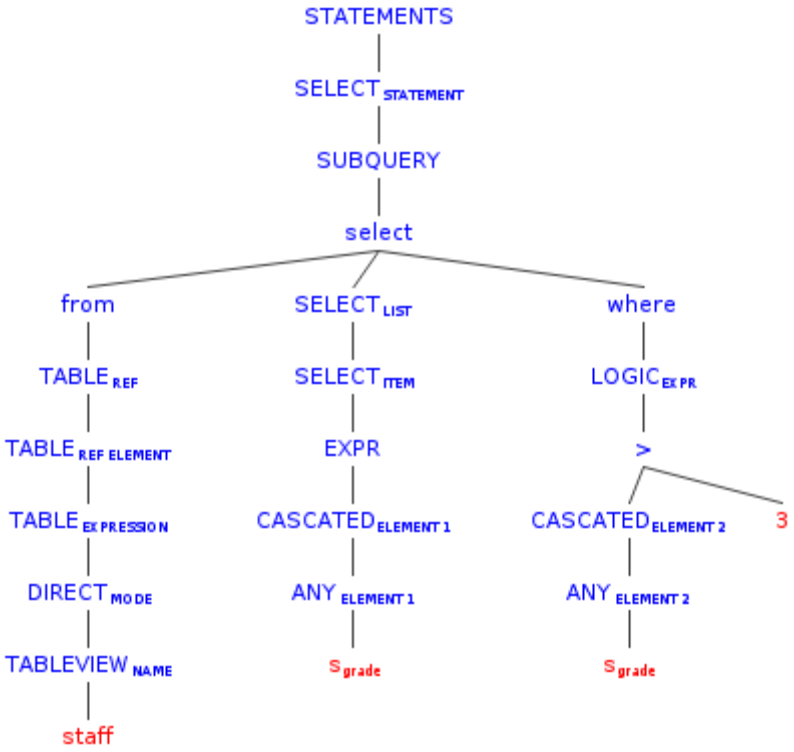


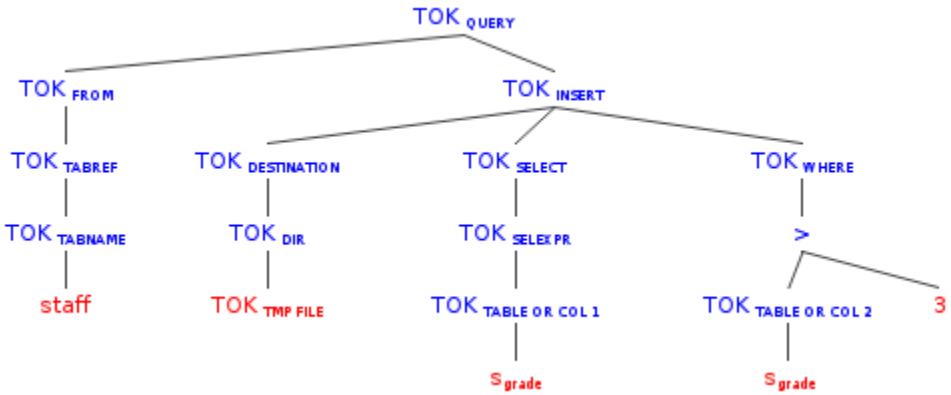
After Transform:



Translater

SQL AST → HIVE AST:





Agenda

Introduce of "Project Panthera ASE"

Design and Architecture

Case study

Performance

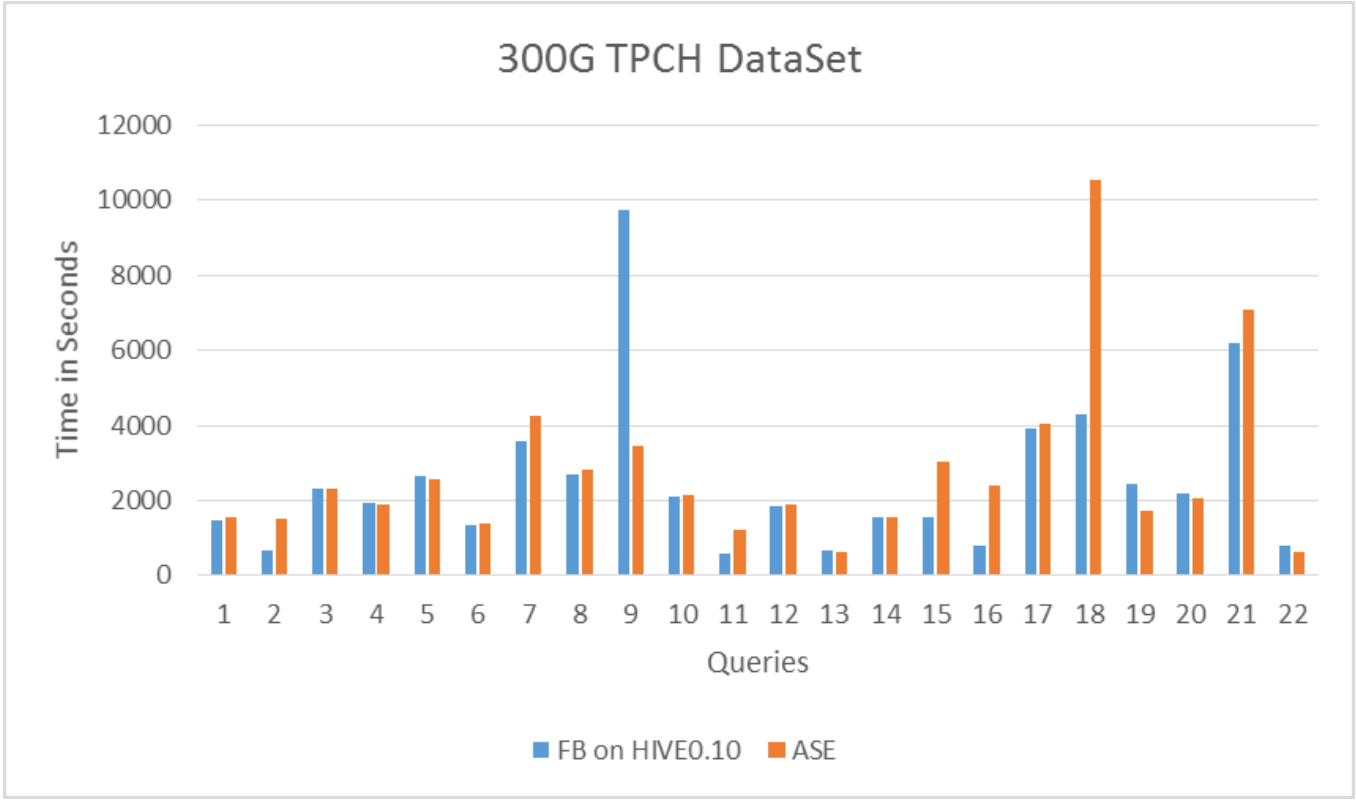
Test Suite

Summary

Panthera ASE performance

HIVE-600 (https://issues.apache.org/jira/browse/HIVE-600)

- Facebook manually transformed TPC-H queries
- Can run directly on HIVE



Agenda

Introduce of "Project Panthera ASE"

Design and Architecture

Case study

Performance

Test Suite

Summary

Test suite

Consists of 1048 DQL queries

- 1026 NIST SQL queries (http://www.itl.nist.gov/div897/ctg/sql_form.htm)
  - A SQL test suite developed jointly by U.S. National Institute of Standards and Technology, National Computing Centre Limited (NCC) in the U.K, and Computer Logic R&D in Greece
  - Help evaluate SQL implementation's conformance, as specified in ANSI X3.135-1992 and ISO/IEC 9075:1992
  - Used among RDBMS (e.g. Oracle, Derby)

- All 22 TPC-H queries (http://www.tpc.org/tpch/)
  - A decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications.
  - The queries and the data populating the database have been chosen to have broad industry-wide relevance.
  - Used by Hive, Impala and etc.

NIST Test

NIST SQL Test Suite Version 6.0

- http://www.itl.nist.gov/div897/ctg/sql_form.htm
- A widely used SQL-92 conformance test suite
- Ported to run under both Hive and the SQL engine
  - SELECT statements only
  - Run against Hive/SQL engine and a RDBMS to verify the results

	Ported Query# From NIST	Hive 0.9		Panthera AES	
		Passed Query#	Pass Rate	Passed Query#	Pass Rate
All queries	1026	777	75.7%	973	94.8%
Subquery related queries	87	0	0%	78	89.7%
Multiple-table select queries	31	0	0%	27	87.1%

Nightly Test Report Sample (Panthera ASE)

Platform	CentOS 6.2
Build Result	Complete
Tests	**125/4217 Failed**
Git Repository	ssh://git-ccr-1.devtools.intel.com:29418/sotc_cloud-hive
Branch	team-dev
Revision	de628310937fab40e411bcfad3e1127785d1e506
NIST Failures	**42 compile errors**

The following result is compared against Last Nightly Report

JUnit Test Summary

Test Group	Tests	Failures	Errors	Success rate	Time (s)
NIST(TPC-H)	22	0	0	100.00%	2451.706
NIST(SubQuery)	87	9	0	89.66%	1543.257
NIST(Multi Table Query)	31	4	0	87.10%	480.142
NIST(All)	1048	53	0	94.94%	9619.403
Hive-SQL92	424	15	0	96.46%	12362.811
Hive	2745 (+882 878 4)	57	0	97.92%	39660.275

detailed info... compare results...

Summary

Panther ASE can perfectly support standard SQL

Nearly no performance loss

Integrated into IDH 2.5, 2.6, 3.1

Open source URL: ***https://github.com/intel-hadoop/project-panthera-ase***


Thank You!
Q&A
Amazing things happen with Intel inside®