Design of Calcite Implicit Type Cast

Background and Motivation

The implicit type coercion is almost supported by every production RDBMS: MYSQL[1], ORACLE[2] and MS-SQL[3], also some Hadoop data warehouse facilitates like HIVE.

As a query optimization engine of many computation engines(i.e. Apache Flink, Apache Drill) and some OLAP engines(like Apache Druid). Calcite would supply better compatibility with sql query to the underlying engines it adapter with if it has the built-in support for implicit type coercion. There are already some JIRA issues that are relative with this topic more or less:

1. CALCITE-2992: Enhance implicit conversions when generating hash join keys for an equiCondition
2. CALCITE-3002: Case statement fails with: SqlValidatorException: Cannot apply '=' to arguments of type '<INTEGER> = <BOOLEAN>'
3. CALCITE-1531: SqlValidatorException when boolean operators are used with NULL
4. CALCITE-3081: Literal NULL should be generated in SqlDialect
5. CALCITE-2829: Use consistent types when processing ranges

Senarios of Type Coercion:

- When data from one object is moved to, compared with, or combined with data from another object, the data may need to be converted from the data type of one object to another.
- When data from a sql result column is moved into a program variable, the data must be converted from the system data type to the data type of the variable.

Current Popular DB's Type Conversion

Oracle Implicit Type Conversion Rules

Oracle Database automatically converts a value from one datatype to another when such a conversion makes sense. Table 2-11 is a matrix of Oracle implicit conversions. The table shows all possible conversions, without regard to the direction of the conversion or the context in which it is made. The rules governing these details follow the table.

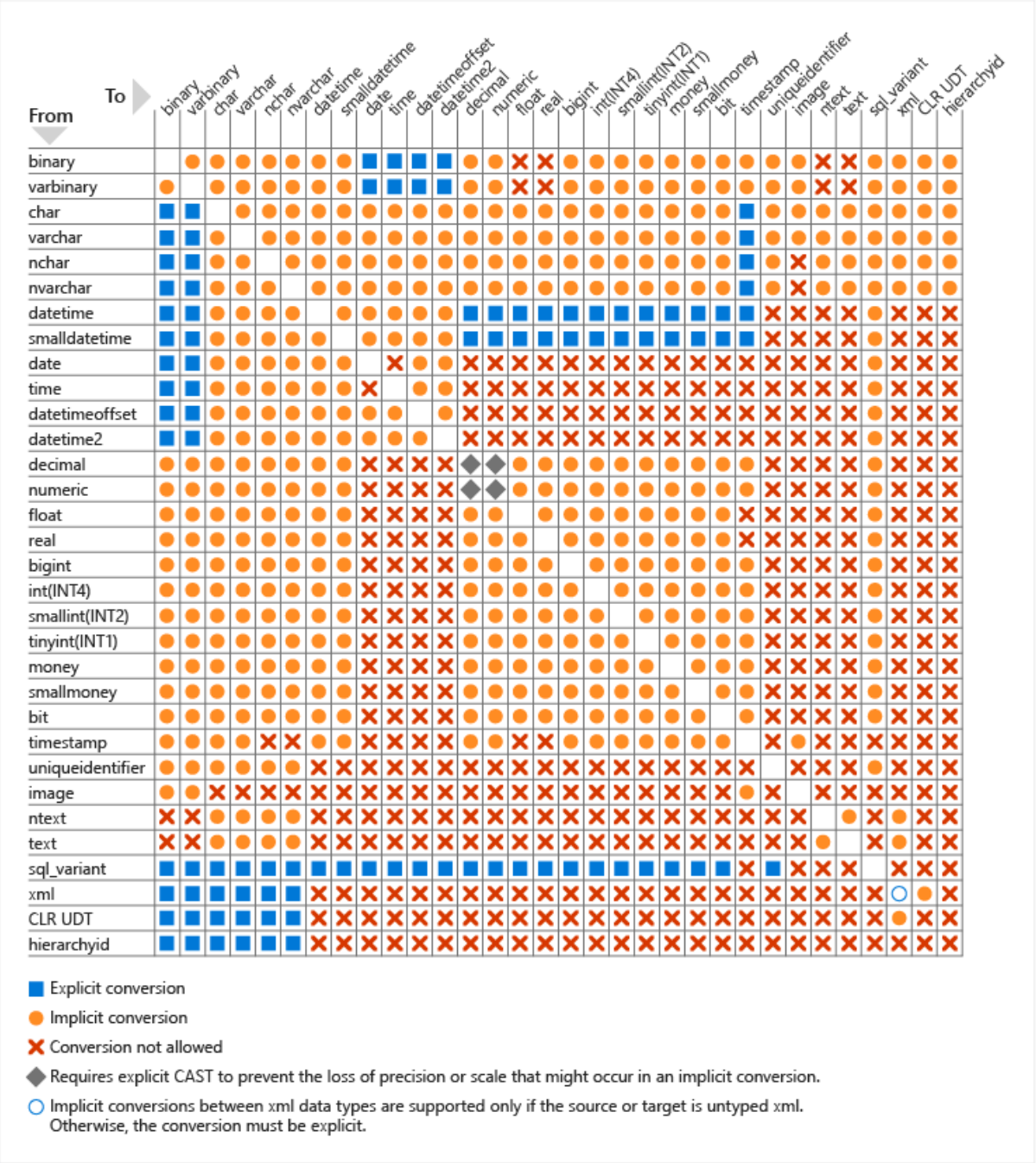
The following rules govern the direction in which Oracle Database makes implicit data type conversions:

- During INSERT and UPDATE operations, Oracle converts the value to the datatype of the affected column.
- During SELECT FROM operations, Oracle converts the data from the column to the type of the target variable.

- When comparing a character value with a numeric value, Oracle converts the character data to a numeric value.
- Conversions between character values or NUMBER values and floating-point number values can be inexact, because the character types and NUMBER use decimal precision to represent the numeric value, and the floating-point numbers use binary precision.
- Conversions from BINARY_FLOAT to BINARY_DOUBLE are exact.
- Conversions from BINARY_DOUBLE to BINARY_FLOAT are inexact if the BINARY_DOUBLE value uses more bits of precision that supported by the BINARY_FLOAT.
- When comparing a character value with a DATE value, Oracle converts the character data to DATE.
- When you use a SQL function or operator with an argument of a data type other than the one it accepts, Oracle converts the argument to the accepted datatype.
- When making assignments, Oracle converts the value on the right side of the equal sign (=) to the datatype of the target of the assignment on the left side.
- During concatenation operations, Oracle converts from noncharacter datatypes to CHAR or NCHAR.
- During arithmetic operations on and comparisons between character and noncharacter datatypes, Oracle converts from any character datatype to a numeric, date, or rowid, as appropriate. In arithmetic operations between CHAR/VARCHAR2 and NCHAR/NVARCHAR2, Oracle converts to a NUMBER.
- Comparisons between CHAR and VARCHAR2 and between NCHAR and NVARCHAR2 types may entail different character sets. The default direction of conversion in such cases is from the database character set to the national character set. Table 2-12 shows the direction of implicit conversions between different character types.
- Most SQL character functions are enabled to accept CLOBs as parameters, and Oracle performs implicit conversions between CLOB and character types. Therefore, functions that are not yet enabled for CLOBs can accept CLOBs through implicit conversion. In such cases, Oracle converts the CLOBs to CHAR or VARCHAR2 before the function is invoked. If the CLOB is larger than 4000 bytes, then Oracle converts only the first 4000 bytes to CHAR.

MS-SQL Implicit Type Conversion Rules

The following illustration shows all explicit and implicit data type conversions that are allowed for SQL Server system-supplied data types. These include xml, bigint, and sql_variant. There is no implicit conversion on assignment from the sql_variant data type, but there is implicit conversion to sql_variant.



[Data type conversion behaviors for MS-SQL]

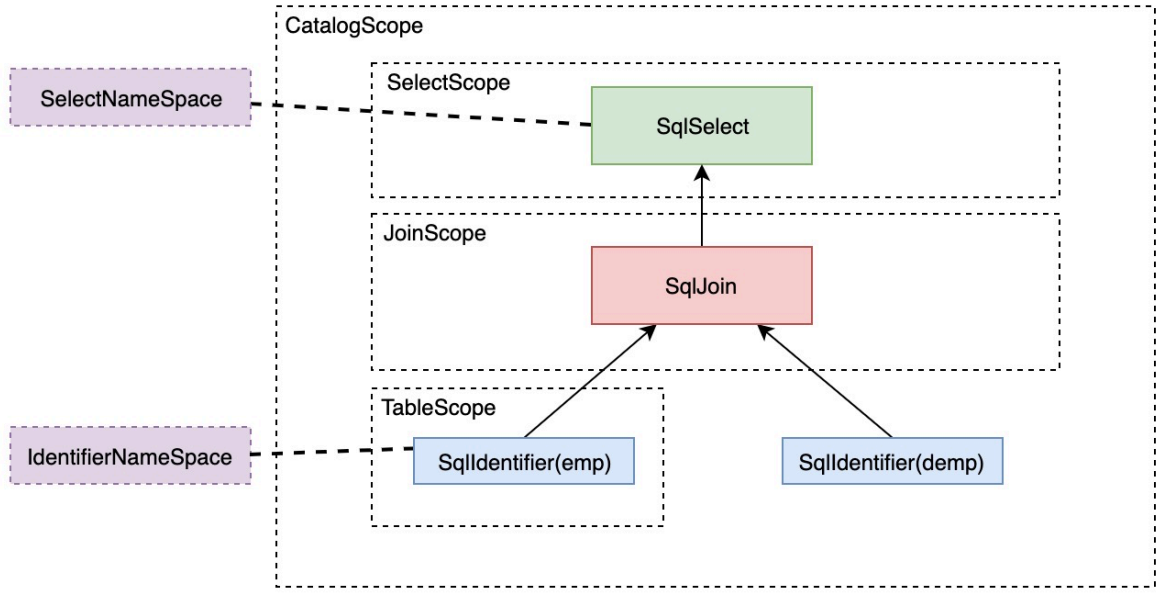
Proposed Design for Calcite

How does Calcite Validate the AST

A sql statement is parsed as a AST (`SqlNode` tree in Calcite) after the invoking of method SqlParser#parseStmt, then it is the `SqlValidator` that deduce the data type of every `SqlNode`. For example, sql statement:

```
select emp.ename, dept.name
from emp
join dept
on emp.deptno = dept.deptno
```

would be parsed to an AST as follows:



`SqlValidator` instantiate the `SqlValidatorScope` for `SqlNode`, i.e. `SqlSelect`, `SqlJoin` and `SqlIdentifier` with order from root node to leaf node. It also registers all kinds of `SqlValidatorNameSpace` for some of the nodes.

`SqlValidatorScope` has Father-son relationship, for AST above, the `CatalogScope` is the parent of `SelectScope`, `JoinScope` and `TableScope`.

`SqlValidatorNameSpace` is the component to lookup the relation row type. `SqlValidator` deduce the relation type from the `IdentifierNameSpace` first then it deduce the node data types in order from leaf node to root node.

A `SqlCall` has 2 kinds of pluggable components to deduce the operands types and the return type. The `SqlOperandTypeChecker` is used to deduce the operands data types while the `SqlReturnTypeInference` is used to deduce the SqlCall's return type.

Implicit type coercion is all about the type of the operands, so the `SqlOperandTypeChecker` is the trigger of our core conversion rules.

Calcite has these operand type checkers:

- FamilyOperandTypeChecker
- CompositeOperandTypeChecker
- SameOperandTypeChecker
- SetopOperandTypeChecker

- ImplicitCastOperandTypeChecker
- AssignableOperandTypeChecker
- ComparableOperandTypeChecker

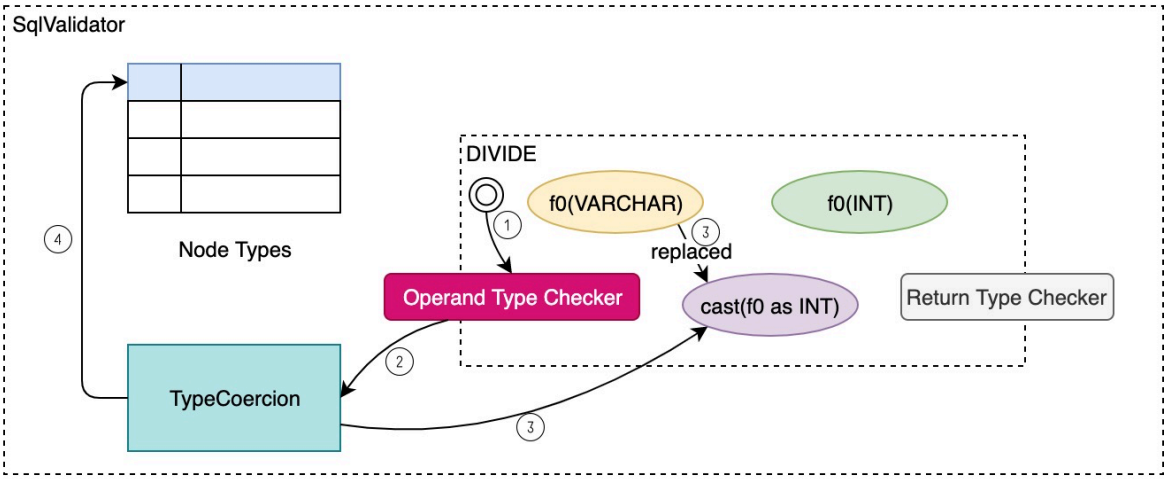
Implicit Type Coercion Work Flow

The validator will check the operands/return types of all kinds of operators:

1. If the validation passes, the validator will just cache the data type (say `RelDataType`) for the `SqlNode` it has validated;
2. If the validation fails, the validator will ask for the `TypeCoercion` component about if we can do an implicit type coercion, if the coercion rules passes, the `TypeCoercion` component will replace the `SqlNode` with a coerced one of desired type (the node may be an operand of an operator or a column of selected row);
3. Then the `TypeCoercion` component would update the inferred type for the casted node and the containing operator/row column type;
4. If the coercion rule fails again, the validator will just throw the exception as is before.

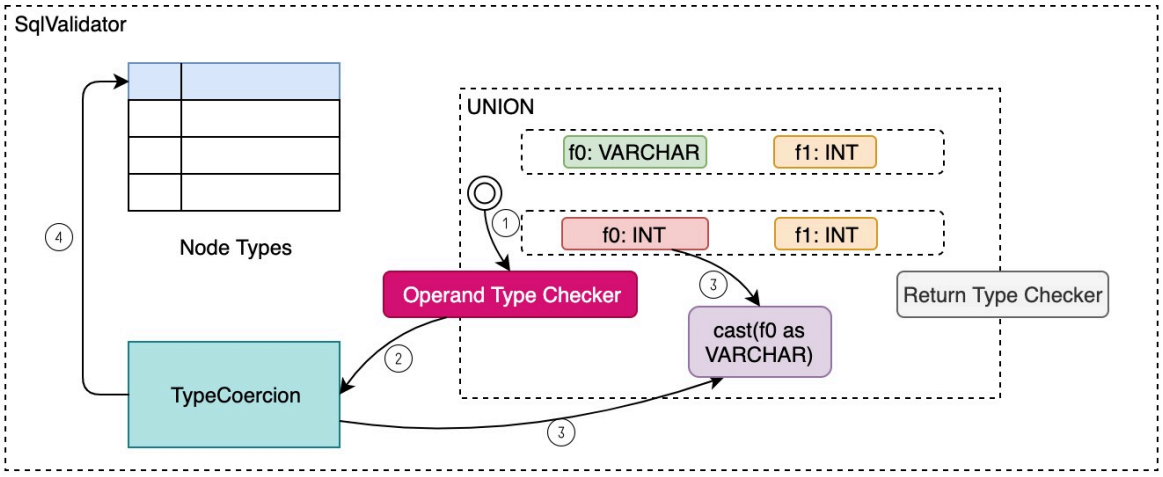
For some cases, although the validation succeed, we still need the type coercion, i.e. for expression 1 > '1', Calcite will just return false without type coercion, we do type coercion eagerly here and the expression would be coerced to 1 > cast('1' as int) whose result evaluates true.

The graph below illustrates how we coerce the operand types of a SqlCall:



1. SqlValidate fires the validation of the call
2. Operand type checker asks the TypeCoercion for the implicit type coercion
3. TypeCoercion wrap the operand with CAST and replace the operand with the new one
4. update node type cache

The graph below illustrates how we coerce the operand type of a SqlNode with struct type:



1. SqlValidate fires the validation of the SET operator
2. Operand type checker asks the TypeCoercion for the implicit type coercion
3. TypeCoercion wrap the column with CAST and replace the column with the new one
4. update node type cache also the row type

Strategies for Finding Common Type

The whole rules to find the proper conversion type:

- If the operator has expected data types, just take them as the desired one. (e.g. the UDF would have eval() method which has reflection argument types)
- If there is no expected data type but the data type families are registered, try to coerce the arguments to the family's default data type, i.e. the String family will have a VARCHAR type.
- If neither expected data type nor families are specified, try to find the tightest common type of the node types, i.e. int and double will return double, the numeric precision does not lose for this case.
- If no tightest common type is found, try to find a wider type, i.e. string and int will return int, we allow some precision loss when widening decimal to fractional, or promote to string type.

We try the best to find the tightest common type that does not lose precision, for example DOUBLE and INT return DOUBLE, DATE and TIMESTAMP return TIMESTAMP; If we can not find the tightest common type, then try to coerce all the operands to VARCHAR type which may lose some precision(or 2 DECIMALs with wider precision/scale).

Conversion Contexts and Strategies

SQL Contexts	Expression subtype	Strategies

Set Operation	union/except/intersect	compare the data type of each branch row to find the common type of each fields pair
Arithmetic Expression	binary arithmetic: [+, -, &, |, ^, /, %, pmod]	1. promote string operand to data type of the other numeric operand; 2. two strings would all be coerced to DECIMAL.
	binary comparison: [=, <, <=, >, >=, <>]	1.promote string and timestamp to timestamp; 2. make 1=true and 0=false always evaluates true; 3. find common type for both operands if there is numeric type operand.
IN Expression	with subquery	compare type of LHS and RHS, find the common type, if it is struct type, find wider type for every field
	without subquery	if RHS is a expr list, compare every expr to find the wider type
Special AGG Function		promote string all to decimal type
Case When Expression	case when expression	find then and else operands common wider type
	Colesce FUNC	same as case when
[Date|Timestamp|String] +/- interval		promote string to timestamp
Function with Expected Inputs Type	builtin functions	look up the families registered in the operand type checker, find the family default type if rule allows it
	UDF/UDAF	try to coerce based on the argument operands types of eval() func

Type Conversion Matrix

The table below illustrates the implicit type coercion rules for all kinds of engines:

From-To	boolean	tinyint	smallint	int	bigint	decimal	float/real	double	interval	date	time	timestamp	[var]char	[var]binary
boolean													s	
tinyint			m s	m s	m s	m s	m s	m s				m	m s	m
smallint		m s		m s	m s	m s	m s	m s				m	m s	m
int		m s	m s		m s	m s	m s	m s				m	m s	m
bigint		m s	m s	m s		m s	m s	m s	o			m	m o s	m
decimal		m s	m s	m s	m s		m s	m s				m	m s	m
float/real		m s	m s	m s	m s	m s		m s					m o s	m
double		m s	m s	m s	m s	m s	m s						m o s	m
interval					o								o	
date												s	m o s	
time													m	
timestamp		m	m	m	m	m				s			m s	m
[var]char		m o s	m o s	m o s	m o s	m o s	m o s	m o s	o	m o s	m	s		s
[var]binary		m	m	m	m	m						m	m s	

c: Calcite	m: MS-SQL	o: Oracle	s: Spark

See CalciteImplicitCasts for the details.

Reference

[1] Mysql type conversion
[2] Oracle Datatype Comparison Rules
[3] SqlServer Data type conversion (Database Engine)