Deploying CEP Technology for Fraud Detection in Financial Transactions

April 25, 2014

George Gettel (gtgettel@wpi.edu)

Devin Roberts (dtroberts@wpi.edu)

Nguyen Tran (nguyent_92@wpi.edu)

Zibo Wang (zibo@wpi.edu)

Advisors: Professor Elke Rundensteiner

Sponsor: ACI Worldwide

TABLE OF CONTENTS

ACKNOWLEDGEMENT

We would like to thank everyone from ACI Worldwide, our sponsoring organization, and WPI project advisors for giving us the opportunity to work on this project and aiding us throughout the project. The progress of the project would not have been possible without their support, insight, and dedication.

Special thanks to:

Professor Elke Rundensteiner

Olga Poppe

Eric Gieseke

Ken Chenis

David Dreyfus

Jeremy Francisco

1 ABSTRACT

Fraud in financial transactions has been under considerable scrutiny lately. Our sponsor, ACI Worldwide, is responsible for processing trillions of credit card transactions every day and their featured product for Risk Management Services (RMS). The RMS uses data from the transaction and the user's profiles to determine if a transaction is fraudulent. To determine this, the RMS requires a database that is very large and continues to grow. However, due to the increasing amount of transaction data and limitations of the SQL language in term of execution speed, ACI is exploring the ability to use Complex Event Processing (CEP) technologies for their next generation product. This project aims to create a model to represent fraud detection queries and an automated process for translating the existing SQL statements of RMS into Esper and Java to utilize the advantages of CEP technology. The major designs of this project include: translating SQL into a XML parse tree, translating the XML parse tree into a Canonical Model, and finally translating the Canonical Model into Esper and Java.

2. Introduction

ACI worldwide produces a Risk Management System (RMS). The product is designed to detect fraud when processing financial transactions. In this section we will define what a financial transaction is in terms of this project, summarize the state of industry of fraud detection, as well as provide a general overview of the Risk Management System currently produced by ACI Worldwide.

2.1 Background on Financial transaction

A financial transaction is a series of electronic messages that describe the exchange of finances. Transactions are initiated when a merchant or user charges a credit account. A transaction event is then sent to the authorization system of a bank, which manages the account. This system confirms that the account has the necessary funds or line of credit. Simultaneously, the transaction event is sent to a financial fraud detection system that will flag the transaction if it is believed to be fraud.

2.2 Financial fraud

2.2.1 Classification

The classification framework of Financial Fraud Detection (FFD) shown in Figure 2.1 is based on the Financial Fraud Detection data mining research, fraud detection research, and the financial crime framework of the U.S. Federal Bureau of Investigation.

The inner ring summarizes the different categories or forms of financial fraud including bank fraud, securities and commodities fraud, insurance fraud, and other related financial fraud. The typical fraudulent activities associated with these categories are shown in Table 2.1.

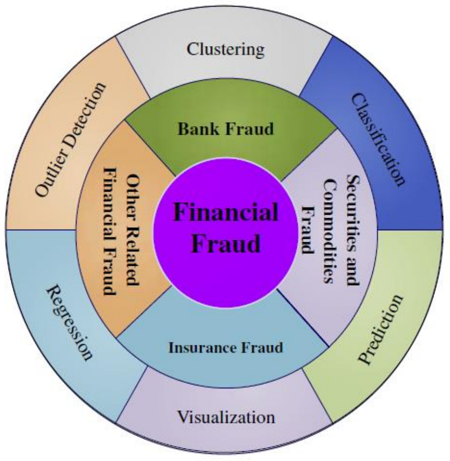


Figure 2.1: Conceptual framework for classifying the applications of data mining for FFD

The outer layer of Figure 2.1 consists of six areas of data mining related to FFD. They are clustering, classification, prediction, visualization, regression, and outlier detection, which are used in fraud detection systems to capture relevant relationships in data.

Bank fraud is defined as “whoever knowingly executes, or attempts to execute, a scheme or artifice (1) to defraud a financial institution; or (2) to obtain any of the moneys, funds, credits, assets, securities, or other property owned by, or under the custody or control of, a financial institution, by means of false or fraudulent pretenses, representations, or promises” by the Federal Deposit Insurance Corporation. Typical activities associated with bank fraud include credit card fraud, money laundering, and mortgage fraud, each with their own various categories.

Insurance fraud is the process of falsifying any point of the insurance process such as, application, eligibility, rating, billing, and claims, and can be committed by anyone involved in the process such as, consumers, agents, healthcare providers, and others. (Gayler, Smith, Lee, and Clifton)

Securities and commodities fraud include market manipulation, high yield investment fraud, the Ponzi scheme, and the pyramid scheme among others, as defined by the FBI. As a typical example of the pyramid scheme, the Ponzi scheme pays returns to its earlier investors with the investment from new investor, but not profit can be earned by actual operation of business.

Financial Fraud Categories	**Fraudulent Activities**
Bank Fraud	Mortgage fraud, Asset forfeiture/money laundering
Insurance Fraud	Healthcare fraud, insurance fraud
Securities and commodities fraud	Securities and commodities fraud
Other related financial fraud	Corporate fraud, Mass marketing fraud

Table 2.1: Financial Fraud Categories and Activities

2.2.2 Fraud in Relation to this Project

This project focuses on software designed to detect credit card fraud, a form of bank fraud. Typical credit card fraud includes unauthorized usages of a card, card forgery, or transaction on an inactive card (Syeda, Zhang, & Pan). These forms of fraud are then observed from the perspective of what is being defrauded, an account holder, the bank, an ATM, etc.

2.2.3 The Fraudster

Clifton Phua et al. define the white-collar fraudster as a manager, employee, and an external party; external party being defined as average offenders, criminal offenders, and organized crime offenders. (Gayler, Smith, Lee, and Clifton) Average offenders participate in random or occasional dishonest behavior with opportunity, or when suffering from financial hardship. Individual criminal offenders and organized/group offenders can be more difficult to detect as they disguise their identities, and change their modus operandi in order to bypass detection systems. Because of this, fraud detection systems must be adaptable to identify new forms of fraud.

2.3 The Scoring Engine and Risk Management System

ACI Worldwide assesses the probability of a fraudulent financial transaction using the Risk Management System product. When a financial transaction is initiated, the transaction event is first sent to the Authorization processing component. This component confirms that the account has the necessary funds, and is otherwise authorized to complete the transaction. Simultaneously, the event is sent to the Risk Management System to determine the probability the transaction is fraudulent.

The RMS consists of two components, the Scoring Engine, and the Rule Engine. The Scoring Engine utilizes a model generated from a neural network. The model is created from transactions that have been verified as legitimate or fraudulent. This model is in n-dimensions where n is the number of processed "features", which we will define in depth, taken from the past transactions. When a new transaction is received, the Scoring Engine plots the transaction using the transaction data and the user's profile in the n-dimensional model. The distance of the transaction from other legitimate and fraudulent transactions in the model is then converted into a score, which represents the statistical probability that the transaction is a case of fraud.

This score is sent to the second component, the Rule Engine, which is a collection of SQL queries run on the database of information that comprised the user’s profile. If these queries return a Boolean value of true, then the transaction is flagged as fraudulent and may or may not be processed at the register. If the result is a false, then the transaction is processed. The Rule Engine is designed to affectively detect current modus operandi of fraudsters, as the banks can update it frequently.

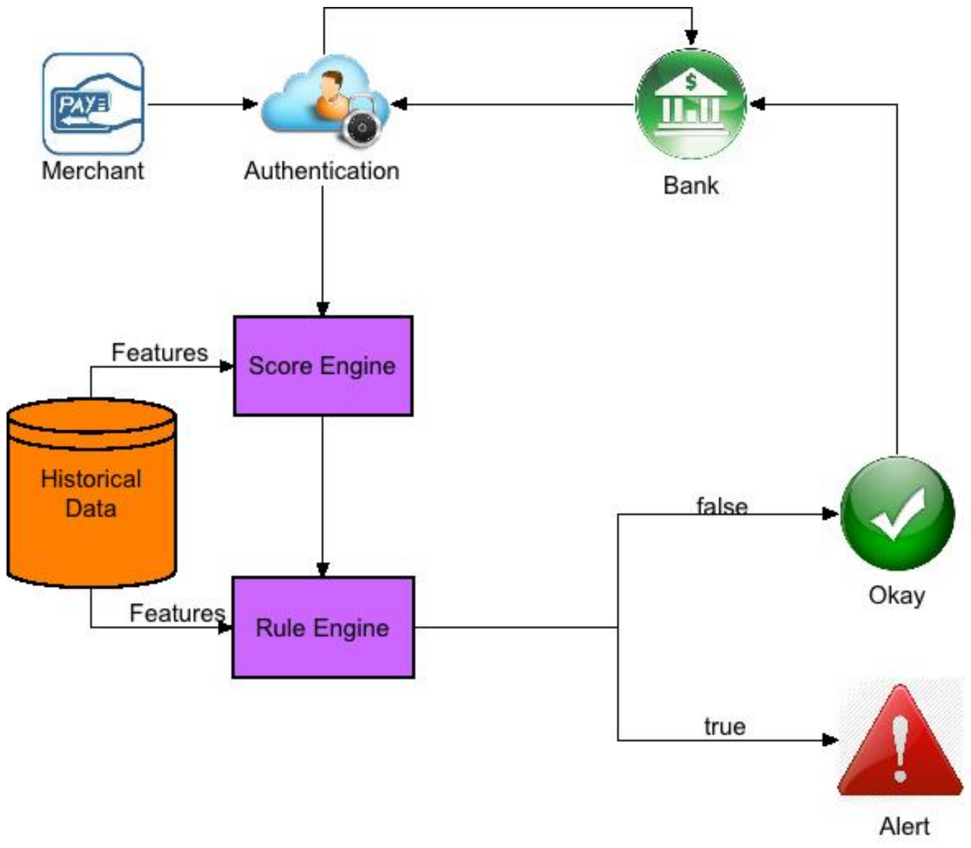


Figure 2.2: Risk Management Service

2.4 LIMITATIONS OF THE CURRENT APPROACH

Currently, the RMS is capable of processing between 2000-2500 transactions per second (TPS). This is limited by the expensive input operations needed to process the queries in the Rule Engine, and to add the features to the Scoring Engine. Because these operations are so expensive the RMS can only process the top 12 features out of 50 in the Scoring Engine, which decreases the accuracy of the model. This also limits the Rule Engine to querying the last 30 days of activity, instead of the profile's entire history.

The system also fails in that as customers create new queries to search for new methods of fraud, the maximum transactions per second decreases. The growth of the Rule Engine increases the processing time faster than new hardware can decrease it.

2.5 Background of Complex Event Processing and Esper

Complex event processing (CEP) delivers high-speed processing of many events across all the layers of an organization, identifying the most meaningful events within the event stream, analyzing their impact, and taking subsequent action in real time.

The engine functions as an inverted database. Instead of storing all of the data from the event streams into a database to then query as SQL does, the system stores the queries in memory, and processes them on events as they arrive in the stream. Processing events in memory and in real-time, instead of storing them in a database to query later, limits expensive input operations and provides a highly scalable architecture (Esper Team and EsperTech Inc.).

Esper was chosen as the CEP engine because of its strong support and design. Esper is an embeddable CEP component written in Java and C# allowing for it to be embedded easily in Java servers, Java virtual machines, or .NET applications. By using Java and C#, events and event attributes can be dynamically typed, and object-oriented design principals can be applied. The developers claim that on a 2GHz Intel CPU, Esper can handle more than 500,000 events per second. While this claim was done on a system tuned for the test and using a stream and event that will not be the same as ACI Worldwide's design, it shows Esper's potential and efficiency (Esper Team and EsperTech Inc.).

2.6 Goals of the Project

The limitations of the RMS's architecture has led ACI Worldwide to experiment with a different architecture for the next iteration, such as using a Complex Event Processing Engine (CEPE), specifically Esper, to replace the time intensive Rule Engine. ACI Worldwide hopes to reach a TPS goal of one million with this new architecture.

Transitioning to this architecture will require customers to stop using the SQL statements they have created for the Rule Engine. This amounts to hundreds of thousands of queries to detect various types of fraud, and significant intellectual property that will no longer be valuable. ACI believes this may prevent their customers from transitioning to the new iteration of the RMS.

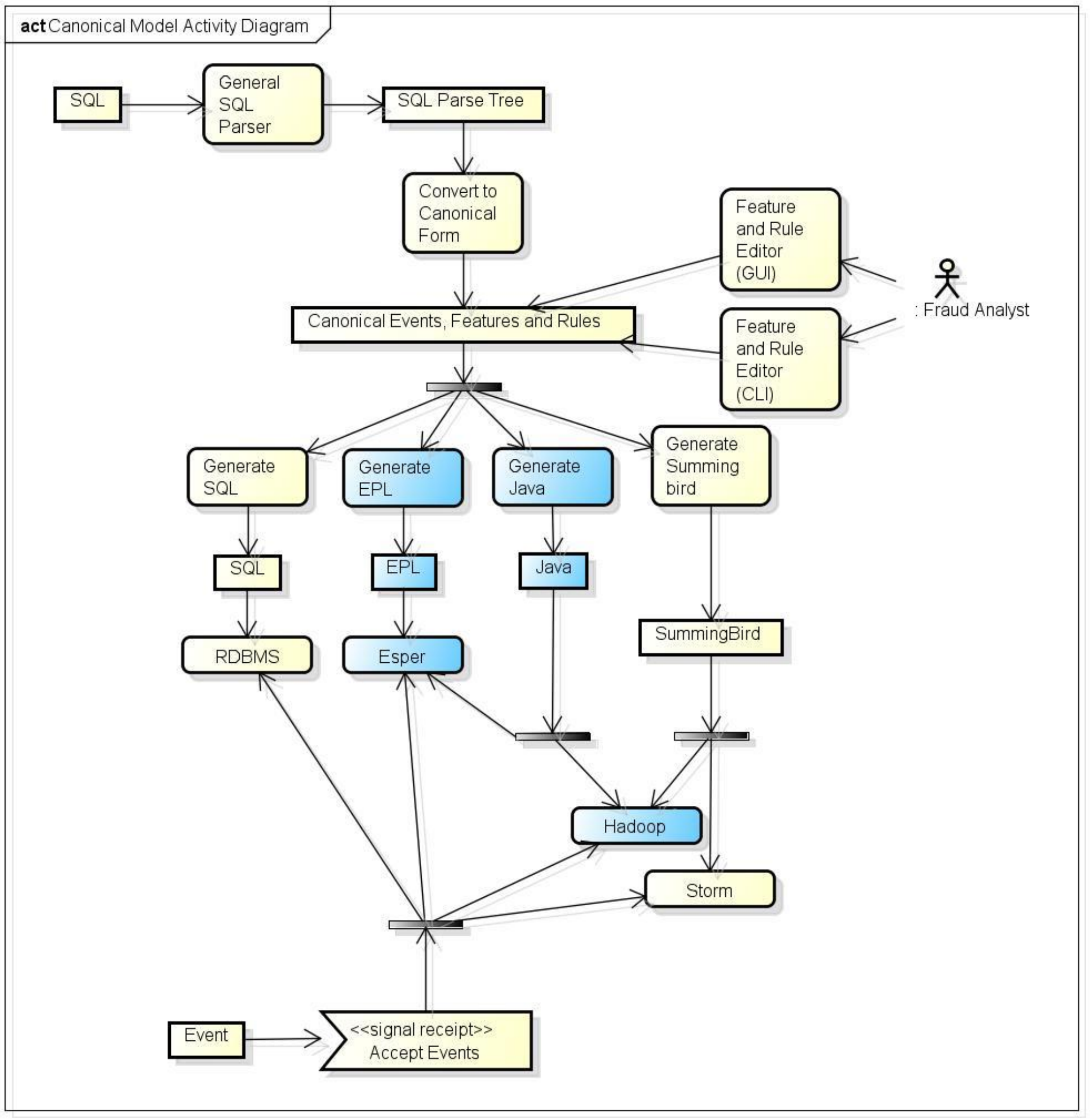

act Canonical Model Activity Diagram
SQL
General SQL Parser
SQL Parse Tree
Convert to Canonical Form
Feature and Rule Editor (GUI)
Feature and Rule Editor (CLI)
: Fraud Analyst
Canonical Events, Features and Rules
Generate SQL
Generate EPL
Generate Java
Generate Summing bird
SQL
EPL
Java
SummingBird
RDBMS
Esper
Hadoop
Storm
Event
<<signal receipt>> Accept Events

Figure 3.1: Project road map

The goal of this project is to produce a canonical model to model these existing fraud rules currently written in SQL and to facilitate the translation of them to other languages. The model will be tested rigorously throught the demonstration of converting SQL to Esper using the intermediary canonical model as shown in Figure 3.1 below. The translation software that will automatically translate SQL queries into Esper event queries, and Java Boolean logic, to facilitate the migration of ACI’s existing customers to the new paradigm. The software must be able to produce a high level canonical form containing the intent of the query, and differentiating between features and rules, queries that gather and evaluate data, rules and features that access the gathered data. This conceptual model's features will then be translated into Esper event queries and its rules will be translated into Java Boolean logic.

3. The Proposed Canonical Model

3.1 The Proposed Canonical Model for Translation

The Canonical Model is by far the most important part of this project. The form serves as the middle step between the original SQL and the final output of either Esper or Java code or any other translation. The canonical form is a representation of the original SQL, the syntax form, and it also includes some derived intention of the original statement's logic, the semantic form. The syntax form is the SQL parse tree of the original statement.

3.1.1 Overview

The Canonical Model is designed to effectively store both the Syntactic and Semantic information of an SQL statement. This means it must be able to store the actual query script, as well as the intent of the query, in an easy to understand data structure.

By storing this information in a Canonical Model, we should then be able to translate any SQL query into any other relational language. In this project, we are focusing on the translation to Esper.

3.1.2 Requirements

The main requirement of the Canonical Model is that it can accurately store the syntactic information in an SQL query. Meaning an instance of the data structure modeled after an SQL query should be able to be translated into that same SQL query with no loss of information.

The other major requirement is the model accurately stores information regarding the intent of the query. For example, if an SQL query is selecting transactions between two *DATETIME* datatypes, the model should clearly represent that the comparison is designed to select a "window" of transactions. This comparison can then easily be translated into an Esper "time window" or other time-based comparison constructs depending on the end translation language.

Finally, the model must be completely independent of SQL or Esper constructs, even though that is our current translation goal. This is so that ACI may choose to translate the model into other systems in the future, without having to modify the model.

3.1.3 Implementation

The model was translated into java classes and objects in the second step of the design process. The following diagram shows the JAVA object structure of the Canonical Model.

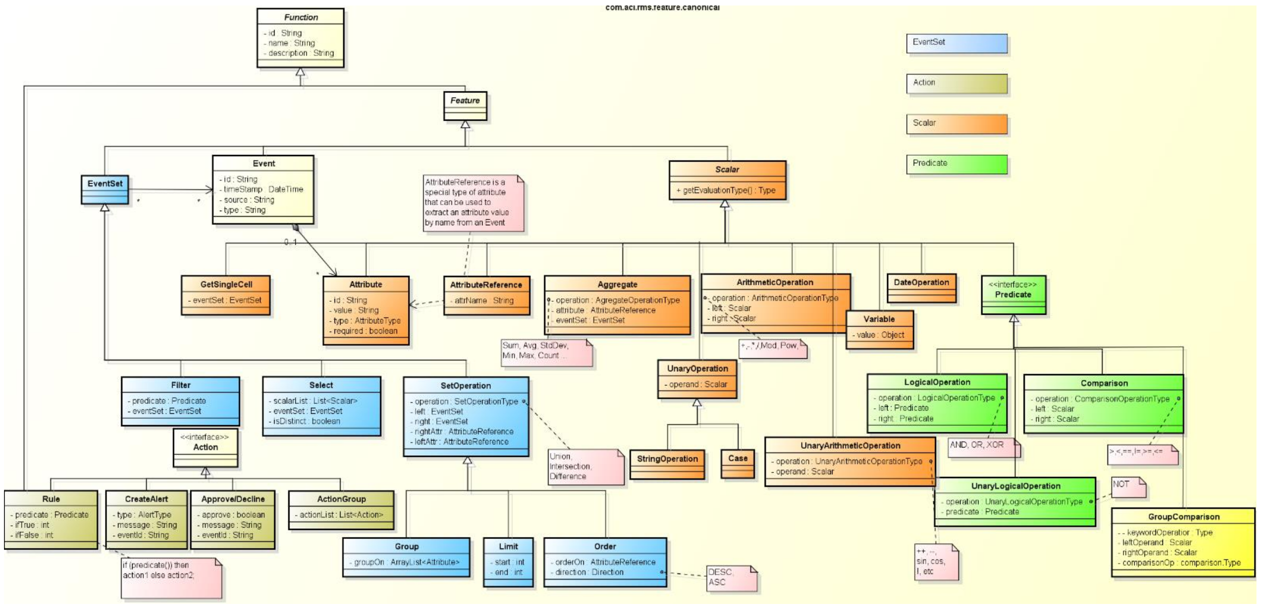


Figure 3.2: Canonical Model for expressing different databasing languages

3.1.4 Implementation Details

Feature instances are created depending on the type of SQL statement being represented. The usual case will be the creation of an *EventSet*, or a Select object, or an Aggregate. These objects will be the roots of a tree comprised of any other objects needed to represent the SQL constructs they are modeled after.

The following example is a SQL query and its corresponding Canonical Model:

Example SQL Query:

```
SELECT COUNT(SD_KEY)
FROM SHORTWINDOW
WHERE @SD_Term_Country = XX
OR @SD_Term_Country = XZ
AND SD_TERM_ID = @SD_TERM_ID
AND @DD_DATE < DD_DATE
AND SD_KEY <> SD_KEY
AND MD_TRAN_AMT1 >= 99  + MD_TRAN_AMT1
```

Model Generating Code:

```
//create the EventSet SHORTWINDOW
EventSet events = new EventSet("Shortwindow");

//create the Attribute sd_key
Attribute at = new Attribute("Shortwindow", "SD_Key");

//create attributes for the sd_term_country, sd_term_id, and the variable sd_term_id
Attribute country = new Attribute("variable", "@SD_Term_Country");
Attribute sd_term_id = new Attribute("shortwindow", "SD_TERM_ID");
Attribute atsd_term_id = new Attribute("variable", "@SD_TERM_ID");

//create variable @dd_date and Attribute dd_date
Attribute atdd_date = new Attribute("variable", "@DD_DATE");
Attribute dd_date = new Attribute("shortwindow", "DD_DATE");

//create other all other Attributes in query.
Attribute md_tran_amt1 = new Attribute("variable", "MD_TRAN_AMT1");
Attribute sd_key = new Attribute("shortwindow", "SD_KEY");
Attribute atsd_key = new Attribute("variable", "SD_KEY");
Attribute md_tran_amtcomp = new Attribute("constant", "99");

//create ArithmeticOperation to represent the + in the query
ArithmeticOperation md_tran_op = new
ArithmeticOperation(ArithmeticOperation.Type.PLUS, md_tran_amtcomp, md_tran_amt1 );

//create comparison objects for the query
Comparison comp1 = new Comparison(Comparison.Type.EQUALS, country, new
Attribute("constant", "XX"));
Comparison comp2 = new Comparison(Comparison.Type.EQUALS, country,  new
Attribute("constant", "XZ"));
Comparison comp3 = new Comparison(Comparison.Type.EQUALS, sd_term_id,
atsd_term_id);
Comparison comp4 = new Comparison(Comparison.Type.LESS, atdd_date, dd_date);
Comparison comp5 = new Comparison(Comparison.Type.NOTEQUALS, atsd_key, sd_key);
Comparison comp6 = new Comparison(Comparison.Type.GREATEREQUALS,
md_tran_amt1, md_tran_op);

//create all logical operators for the query
LogicalOperation op1 = new LogicalOperation(LogicalOperation.Type.OR, comp1, comp2);
LogicalOperation op2 = new LogicalOperation(LogicalOperation.Type.AND,op1, comp3);
LogicalOperation op3 = new LogicalOperation(LogicalOperation.Type.AND,op2, comp4);
LogicalOperation op4 = new LogicalOperation(LogicalOperation.Type.AND,op3, comp5);
LogicalOperation op5 = new LogicalOperation(LogicalOperation.Type.AND,op4, comp6);

//create filter object to represent entire WHERE clause, and consequently all comparisons
and logical operators
Filter filter = new Filter(op5, events);

//create aggregate to serve as the root of the model instance for this query
Aggregate count = new Aggregate(Aggregate.Type.ALL, filter, at);
```

4. Translation Strategies

4.1 Strategy for Translation from SQL to Canonical Model

4.1.1 From SQL to XML parse tree

The first step in the process of converting the SQL queries into Esper and Java code is to translate the queries into the canonical form, which will serve as the middle step between the original SQL and Esper, Java, or any other translation.

Before translating into the canonical form, we wanted to be able to parse SQL to a XML representation. The XML representation of the original SQL is actually the syntax form of SQL. The form also includes semantic information derived from the SQL statement, such as the intention and logic.

Due to time limitation, we decided to look for a commercial SQL Parser that we could purchase and then modify. We considered the following options: General SQL Parser, ZQL, ANTLR, PHP SQL Parser, and the DMS Software Reengineering Toolkit. Of these choices, the General SQL Parser met our requirements and specifications the best.

Going from SQL, the General SQL Parser (GSP) gives us an extensive parse tree that contains all of the syntactical information in an easy to extract format. Furthermore, if information is missing, then we can add it directly to the parser. From the parse tree, we will go to a canonical form from which we could convert to any database management language. This model is intended to make a query's intent clear while also using lower-level constructs which can easily be converted into other languages.

4.1.2 General SQL parser output

The following section will show an example query provided by ACI Worldwide, and the output produced when processed by the GSP.

```
(SELECT COUNT(DISTINCT(SD_KEY))+1
FROM SHORTWINDOW PAST (NOLOCK)
WHERE @SD_TERM_ID = SHORTWINDOW.SD_TERM_ID
AND @DD_DATE < SHORTWINDOW.DD_DATE
AND @DD_DATE > DATEADD(MI, -10, SHORTWINDOW.DD_DATE)
AND MD_TRAN_AMT1 >= 100
AND @SD_KEY <> SHORTWINDOW.SD_KEY
AND @SD_TERM_CNTRY IN ('XX','XZ'))
```

Statement 4.1: SQL Example Query

The above query is returning the number of distinct transactions from a specific terminal (@SD_TERM_ID) in a range of the past 10 minutes, if the terminal is in either country labeled ‘XX’ or ‘XZ’, and the amount of the transaction is greater or equal to 100. The first part of the SQL statement is:

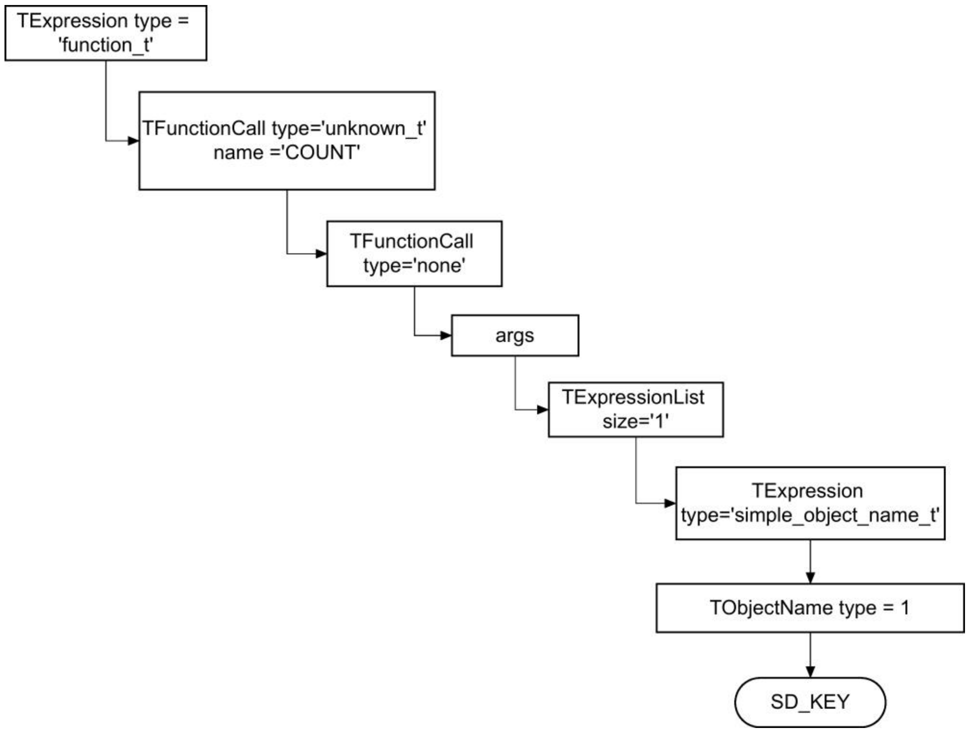
```
SELECT COUNT(SD_KEY)
```

Code Fragment 4.1: SELECT clause from Statement 3.1

If Statement 1 is broken down, the first part is the SELECT statement. This is represented in the XML by the *TSelectSqlStatement* tag. In fact, this statement has a SELECT statement that is not important: the returned object and its type is. The first important part of Statement 4.1 is the COUNT function named TFunctionCall in XML 4.1. The *TFunctionCall* tag is used for any SQL functions in the statement.

In the XML parse trees below, the ellipse shape represents the SQL statement and the rectangle shape represents their corresponding tags. A diamond shape is an operator, which usually operates on two other values. A downward and right-leading edge means the tag is in the sublevel of its upper tag, while a directly downward edge means the tag is in the same level of its upper-tag.

The parse tree for this COUNT is:


TExpression type = 'function_t'
TFunctionCall type='unknown_t' name ='COUNT'
TFunctionCall type='none'
args
TExpressionList size='1'
TExpression type='simple_object_name_t'
TObjectName type = 1
SD_KEY

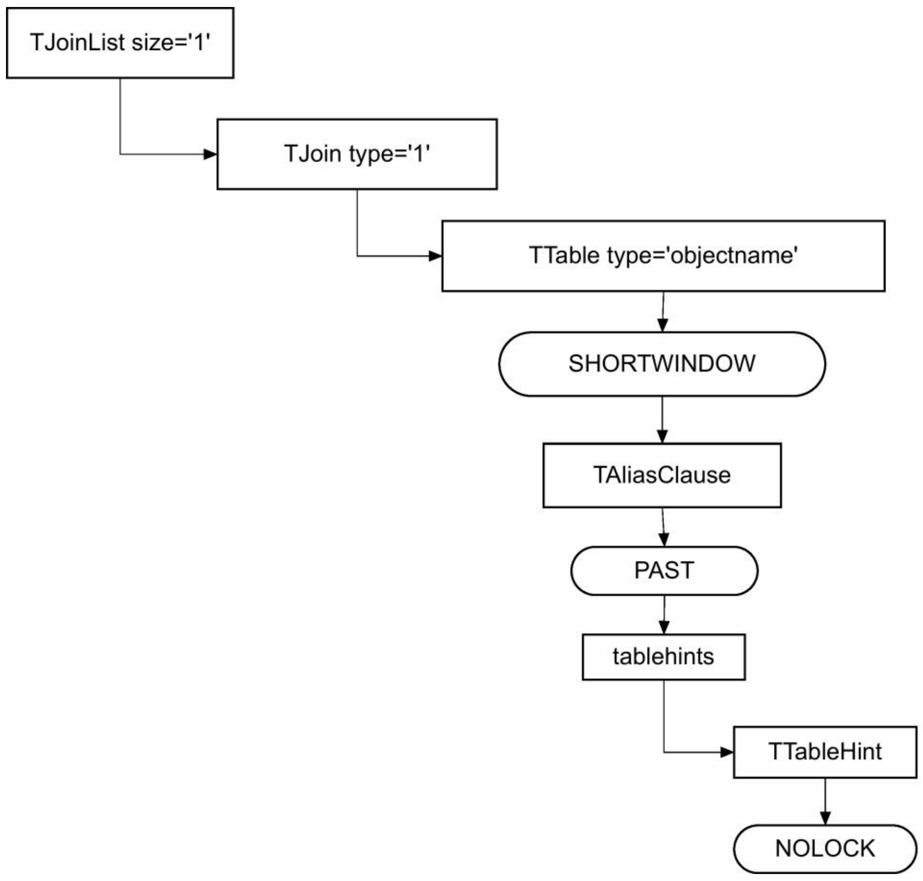
XML 4.1: XML parse tree representation of the COUNT function from Statement 4.1

The next part of Statement 4.1 is the FROM clause:

```
FROM SHORTWINDOW PAST (NOLOCK)
```

Code Fragment 4.2: FROM clause from Statement 4.1

In XML 2.1, this is represented by the *TJoin* branch. This statement has a very simple clause that consists of only one table, *SHORTWINDOW*. The XML looks like:



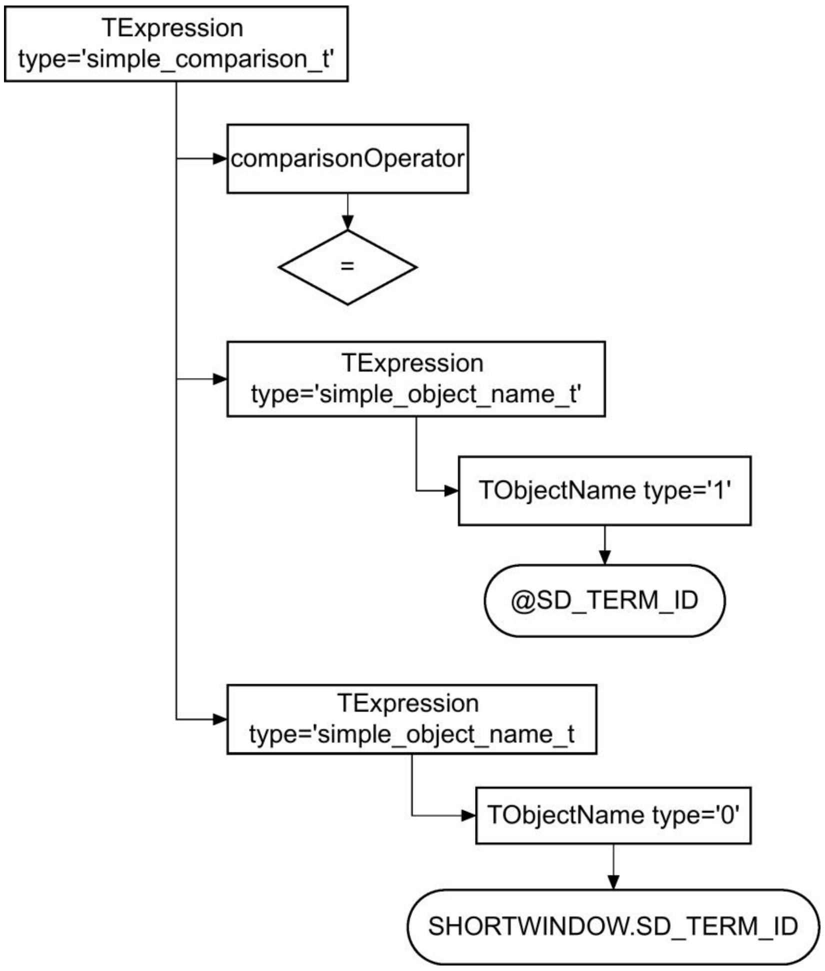
XML 4.2: XML parse tree representation of the *FROM* clause from statement 4.1

The *tablehints* tag tells the database not to lock the table in this case so that more queries can be run simultaneously improving performance. This part is irrelevant to our translation, but it is still part of the parse tree. The final clause of Statement 4.1 is the *WHERE* clause:

```
WHERE @SD_TERM_ID = SHORTWINDOW.SD_TERM_ID
AND @DD_DATE < SHORTWINDOW.DD_DATE
AND @DD_DATE > DATEADD(MI, -10, SHORTWINDOW.DD_DATE)
```

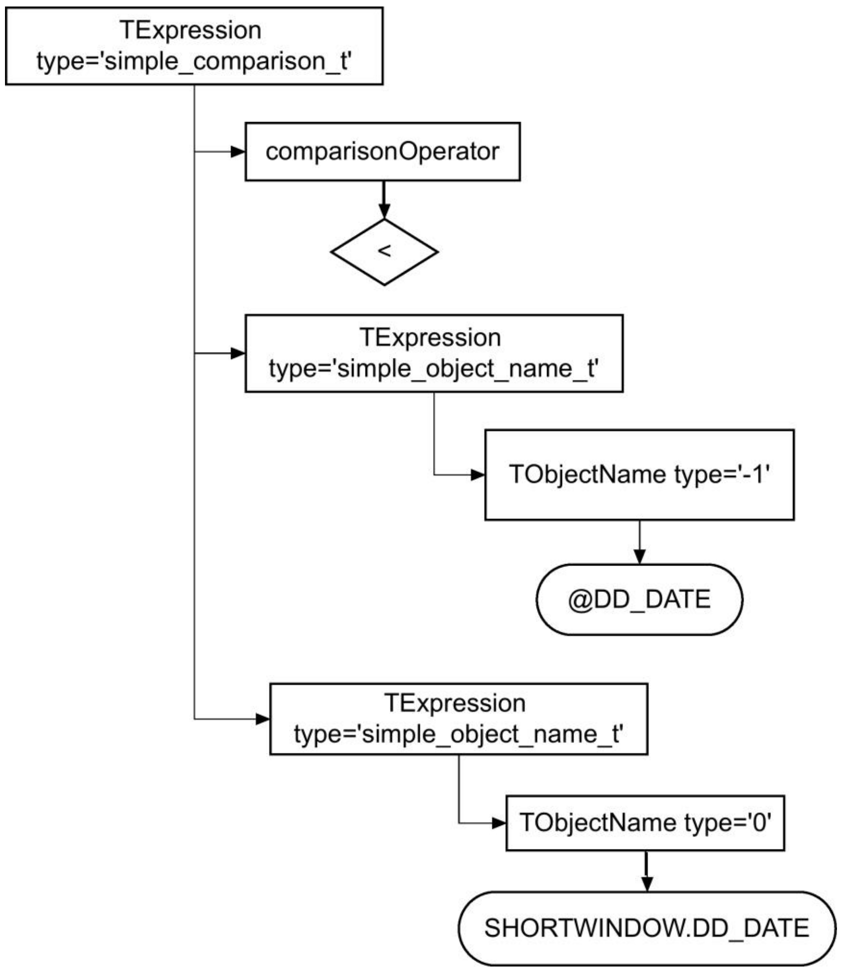
Code Fragment 4.3: WHERE clause from Statement 4.1

The clause is represented by the *TWhereClause* tag in the XML. The first line is a simple comparison and translated into:


TExpression type='simple_comparison_t'
comparisonOperator
=
TExpression type='simple_object_name_t'
TObjectName type='1'
@SD_TERM_ID
TExpression type='simple_object_name_t
TObjectName type='0'
SHORTWINDOW.SD_TERM_ID

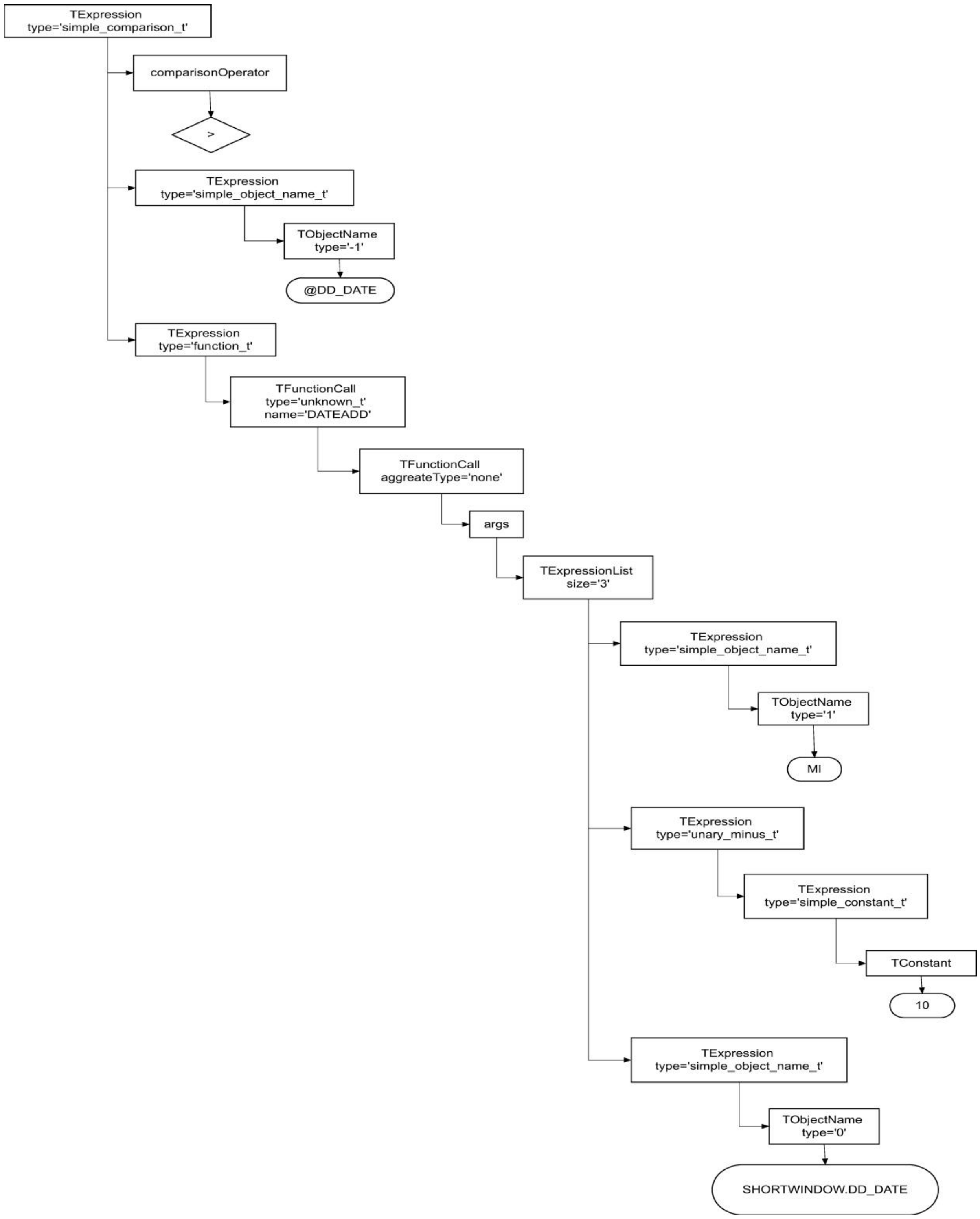
XML 4.3: XML parse tree for the first line of the *WHERE* clause from Statement 4.1

The second line is just as simple and translates to:

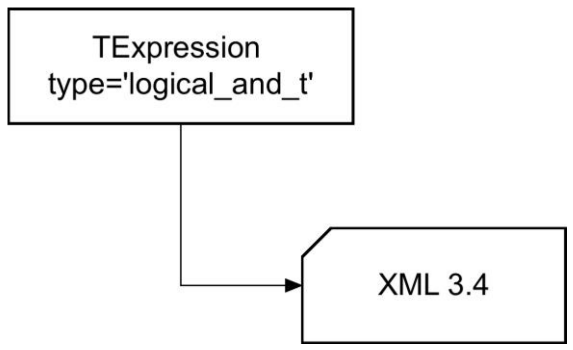

TExpression type='simple_comparison_t'
comparisonOperator
<
TExpression type='simple_object_name_t'
TObjectName type='-1'
@DD_DATE
TExpression type='simple_object_name_t'
TObjectName type='0'
SHORTWINDOW.DD_DATE

XML 4.4: XML parse tree for the second line of the WHERE clause from Statement 4.1

The third line is more complicated because it includes a *TFunctionCall*, *DATEADD*:

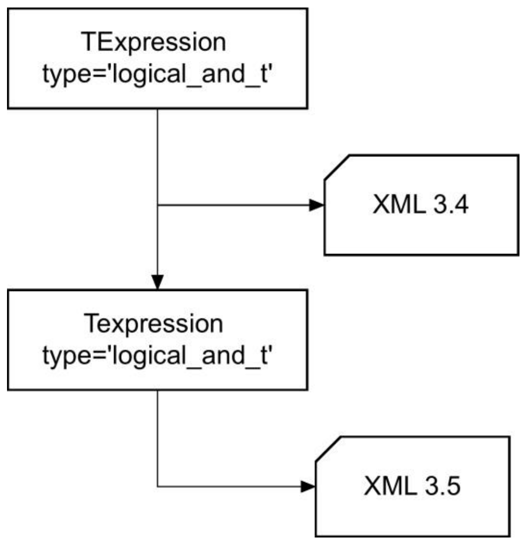

TExpression type='simple_comparison_t'
comparisonOperator
>
TExpression type='simple_object_name_t'
TObjectName type='-1'
@DD_DATE
TExpression type='function_t'
TFunctionCall type='unknown_t' name='DATEADD'
TFunctionCall aggreateType='none'
args
TExpressionList size='3'
TExpression type='simple_object_name_t'
TObjectName type='1'
MI
TExpression type='unary_minus_t'
TExpression type='simple_constant_t'
TConstant
10
TExpression type='simple_object_name_t'
TObjectName type='0'
SHORTWINDOW.DD_DATE

XML 4.5: XML parse tree for the third line of Statement 4.1



XML 4.6: XML parse tree for AND functions

These *AND* functions are nested such that the parse tree for statement 4.1 looks like:



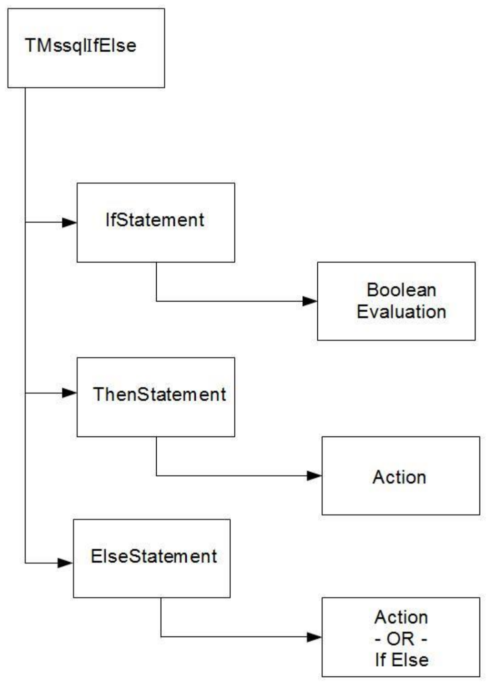
XML 4.7: XML parse tree for nested *AND* functions

Rules at its simplest form are IF... ELSE statements. Consider the rule:

```
        IF (SELECT COUNT(DISTINCT(SD_KEY))+1
                FROM SHORTWINDOW PAST (NOLOCK)
                WHERE @SD_TERM_ID = SHORTWINDOW.SD_TERM_ID
                AND @DD_DATE < SHORTWINDOW.DD_DATE
                AND @DD_DATE > DATEADD(MI, -10, SHORTWINDOW.DD_DATE)
                AND MD_TRAN_AMT1 >= 100
                AND @SD_KEY <> SHORTWINDOW.SD_KEY
                AND @SD_TERM_CNTRY IN ('XX','XZ')) > 100
              PRINT 'approved'
        ELSE
              PRINT 'declined'
```

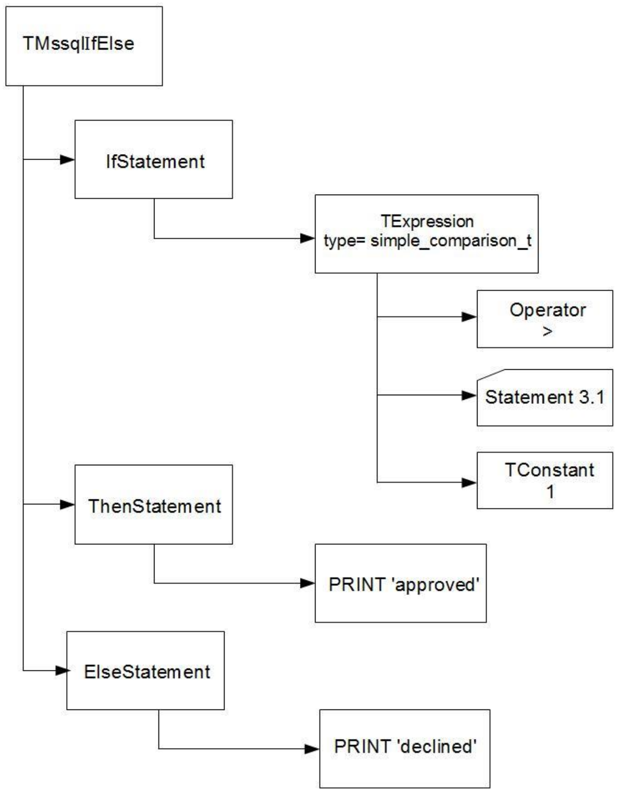
Statement 4.2: Example Rule in SQL

This rule is composed of a simple comparison using the Statement 4.1. The XML for Statement 4.1 remains as it is defined above, but it is wrapped in, for MSSQL, a TMsqlIfElse tag. Within that tag, the IF comparison is contained in an IfStatement tag and the then action is contained in a ThenStatement. The ELSE action is contained within the TMsqlIfElse tag below the IfStatement tag in an ElseStatement tag, which contains the else action. While this action does not contain an IF ELSE statement, it is very possible for one to occur. If there is an IF ELSE, it is composed like the IF... ELSE statement mentioned above with an IfStatement tag representing the ELSE IF statement, a ThenStatement tag representing the IF ELSE statement's action, and an ElseStatement tag representing the ELSE statement. However, the IfStatement tags for the ELSE IF statement is nested inside the parent rule statement's ElseStatement tag.



XML 4.8: XML parse tree of MSSQL IF... ELSE statement

The example in Statement 4.2 is a very simple MSSQL IF... ELSE and it fits the outline of the parse tree in XML 4.8. When filled with values, it is:



XML 4.9: XML parse tree of Statement 4.2

Very simply, within the IfStatement tags, the smaller rule, the comparison, that is comprised of the operator, Statement 4.1, and '1'. Statement 4.1 is the feature of the rule. The action of the IF statement is contained in the ThenStatement tag. In the example, the THEN action is the PRINT 'approved'. The ElseStatement in this example contains the ELSE action, which is PRINT 'declined'. If there was any ELSE IF or sub-IF, it would be parsed into the ElseStatement tag.

4.1.3 From XML to canonical model

The next step is to convert the parse tree that is held in an XML representation to the Java canonical model. For this, certain parts of the XML are mapped to the model objects. To explain this process, the sample query in Statement 4.2, which is a simplified version of Statement 4.1, will be converted to the necessary Java objects. For the purpose of explaining this conversion, it is simpler to consider the actual SQL statement as the XML representation is at the same computational level and is much larger.

```
        SELECT COUNT(DISTINCT (SD_KEY))+1
        FROM SHORTWINDOW
        WHERE @SD_TERM_ID = SHORTWINDOW.SD_TERM_ID
        AND @DD_DATE < SHORTWINDOW.DD_DATE
```

Statement 4.3: SQL Example Query

The query in Statement 3.2 is a *SELECT* statement with an arithmetic operation. Therefore, the top-level model object is an *ArithmeticOperation* object with an operation of '+', a right operand of '1', and a left operand referencing the aggregate function. In the case of the query, the aggregate function counts the number of distinct '*SD_KEY'* values. Therefore, the *AGGREGATE* object takes an *EventSet* of type *Distinct*, which has a Scalar object *Attribute* with a value of '*SD_KEY'* and another *EventSet* object. The EventSet of the *DISTINCT* object references the *WHERE* statement of the SQL query.

The second part of the *DISTINCT* object is the *EventSet* object. In this case, it is a filter object. The filter references a predicate object and another *EventSet*. The predicate refers to the XML representation of the *WHERE* clause of the SQL statement. The predicate is composed of two left and right Comparison object, which correspond to the two comparisons in the *WHERE* clause. The first Comparison has an operator of '=' and two Scalar objects of type *Attribute*. One corresponds to '*@SD_TERM_ID'* and the other corresponds to *'SHORTWINDOW.SD_TERM_ID'*. The second Comparison object has an operator of '<' and two Scalar objects of type *Attribute*. One corresponds to '*@DD_DATE'* and the other corresponds to *'SHORTWINDOW.DD_DATE'*. The *EventSet* of the filter corresponds to the *FROM* clause of the SQL query. Here, it is represented as an *EventSet* with a name of *'SHORTWINDOW'*.

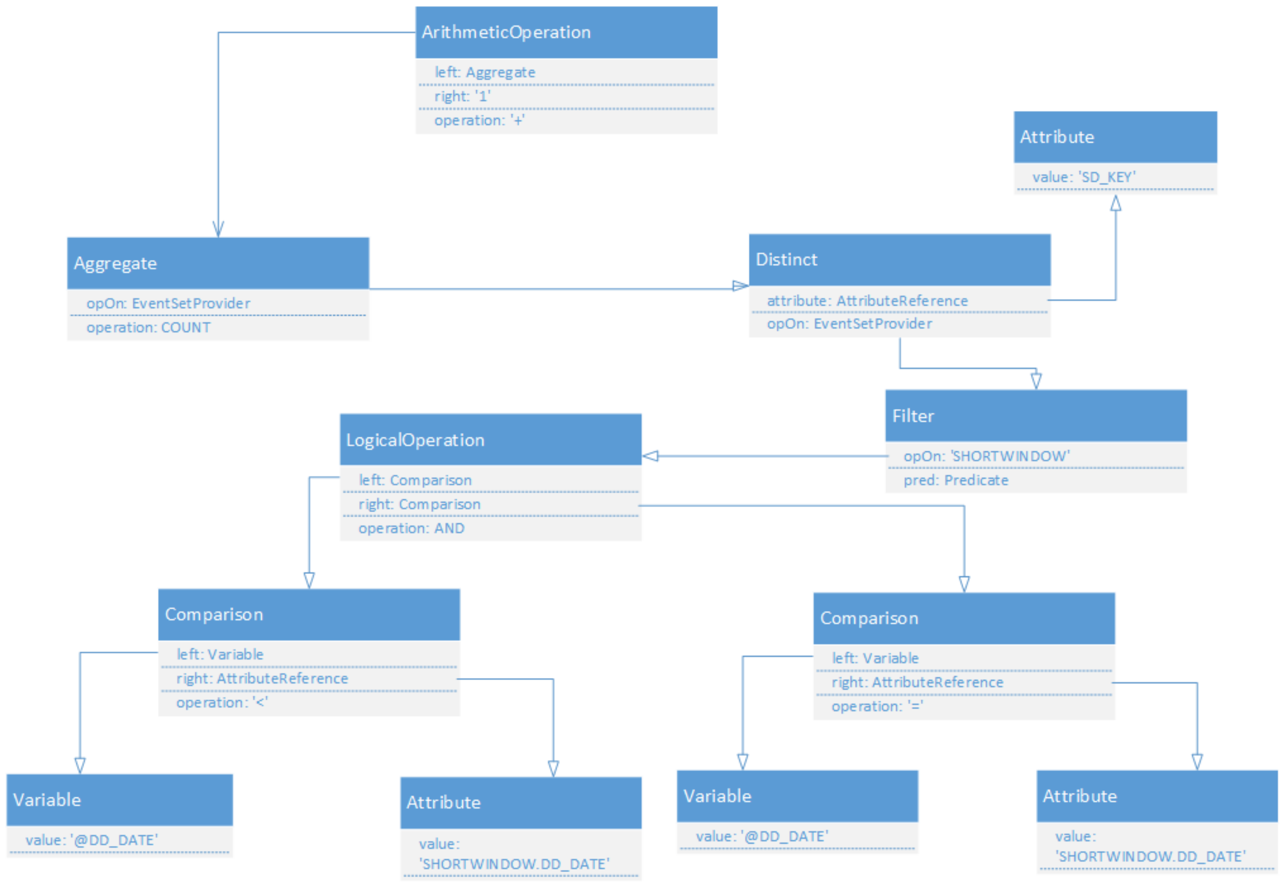


Figure 4.3: Java object representation of Statement 4.3

Below is another example of the translation. Consider the following SQL statement.

```
SELECT SHORTWINDOW.DD_DATE, COUNT(CUST_TBL.PURCHASES)
FROM SHORTWINDOW  INTERSECT CUST_TBL
GROUP BY DD_DATE
HAVING AVG(TRANS_AMT) >= 1000;
```

Statement 4.4: SQL sample query.

This query uses an intersection to select the date and number of transactions greater than $1000 from two separate tables. It is translated into the following object representation.

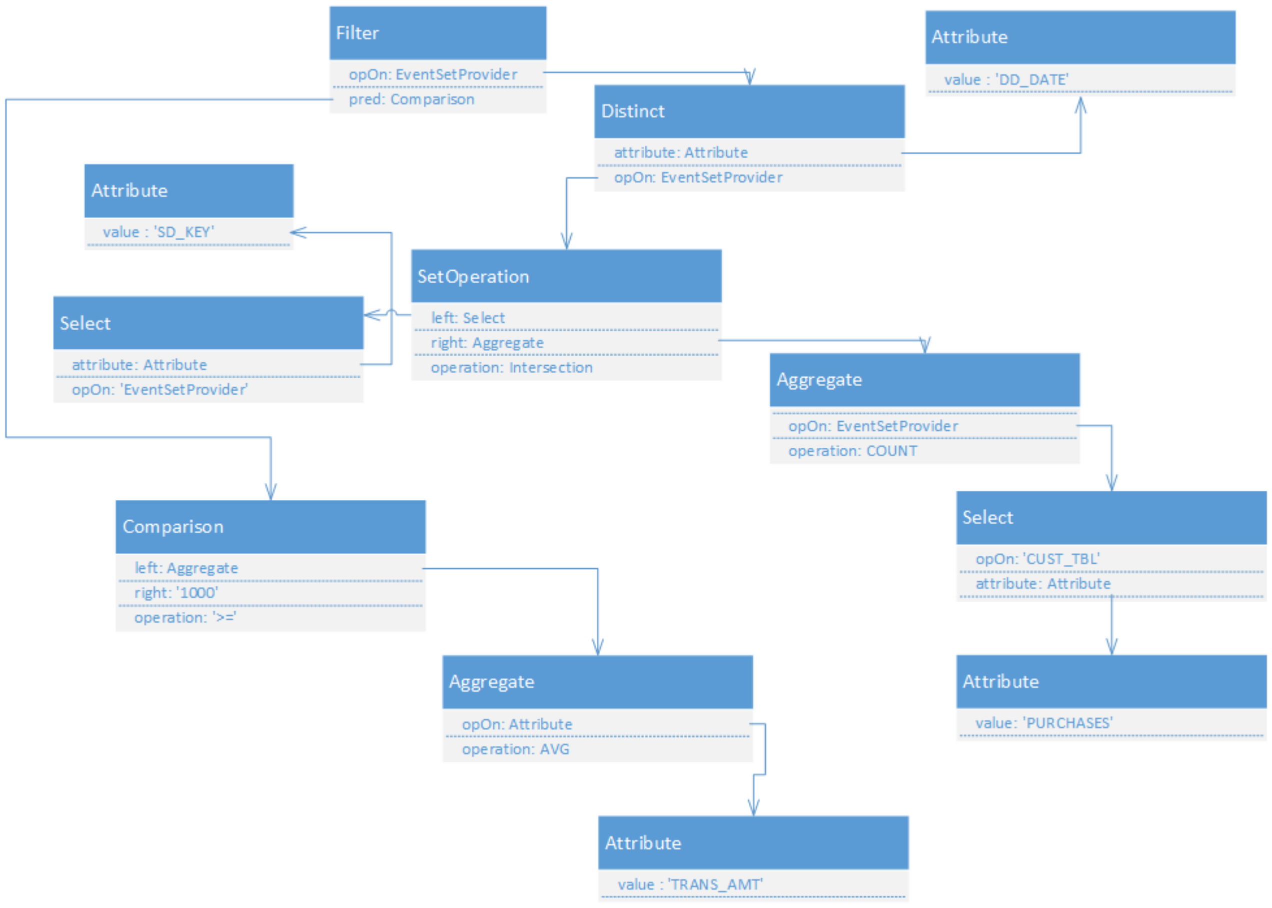


Figure 4.4: Java object representation of Statement 4.4.

For rules, IF… ELSE statements, the canonical model's main object for defining rules is the *Action* interface. This interface is extended by two important objects, *Rule* object and *CreateAlert* object. The *Rule* object contains a predicate, which is used to define the IF statement in the same way that the predicate defines the *WHERE* clause of an SQL statement. The *Rule* object also takes an *ifTrue* and *ifFalse Action*. Therefore, the *THEN* action of the IF statement is in the *ifTrue* and can be a *Rule* if it is a nested IF statement or a *CreateAlert* if it is a print statement.

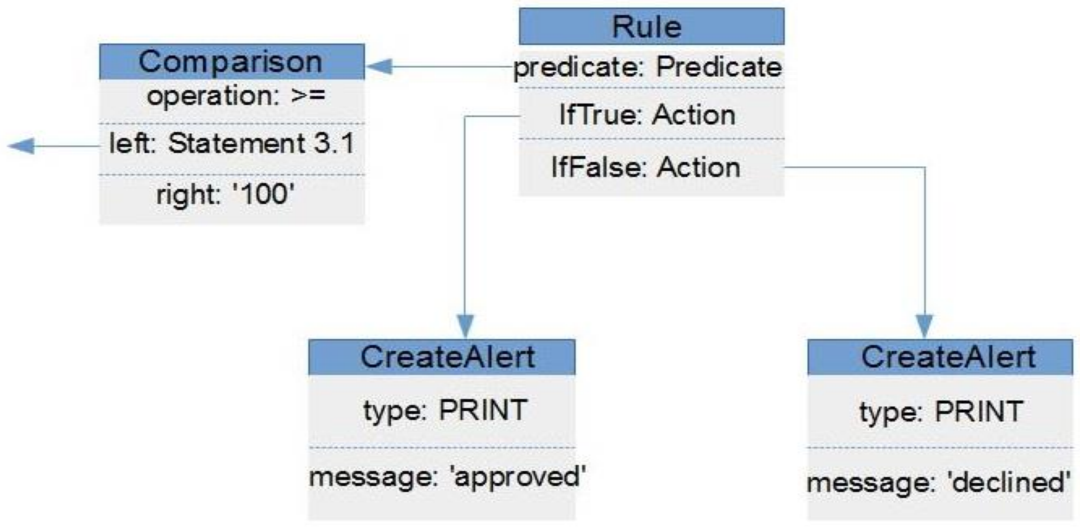


Figure 4.5: Java object representation of rule Statement 4.2

As shown in Figure 4.5, the pattern for rules and alerts is quite simple and easy to repeat. Nested queries are able to retain their properties and hierarchy with this pattern.

4.2 Strategy for Translation from The Canonical Model To EPL and Java

Once we have the canonical representation of SQL statement, the next step is to convert it into Esper. Esper is an Event Stream Processing (ESP) and event correlation engine. Esper's implementation of Event Processing Language (EPL) maintains many SQL-like language constructs that supports many equivalent keywords and hierarchy of SQL, such as *SELECT*, *FROM*, *WHERE*, *GROUP BY*, *HAVING*, *ORDER BY* clauses. The most significant different between Esper and SQL is that Esper works on both event streams and traditional database relations.

4.2.1 Esper Overview

4.2.1.1 Equivalent language constructs of SQL and Esper

As mentioned above, the Esper's implementation of EPL has many equivalent expressions as SQL, both in syntax and functionality. The equivalent statements are listed below.

- Select
- Distinct
- From
- As
- Where
- * (all)
- In
- Between
- And/ or / not
- Having
- Functions
  - Sum
  - Avg
  - Count
  - Group by/Order by
- Operators
  - Equal: =
  - Not equal: !=, <>
  - Comparison operators: <, >, >=, <=
  - Arithmetic operators: +, -, *, /
- First/last
- Exists
- Max/min
- Like
- Join
  - Join

- Cross join
- Full/Left/Right
- Inner(default)/Outer
- On

4.2.1.2 Unique language constructs of Esper

Besides the above list of equivalent statements, Esper also provides a lot of unique features that can be used to identify fraudulent transactions. Four typical aspects of Esper are Pattern, Output, Recognize, and Window. In terms of this project, our primary concern of the unique language constructs of Esper will be focused on Esper time window.

Esper Window

One key feature that enables Esper to execute faster is the named window, as well as the window view functions. The named windows are data windows that allow to be inserted-into or deleted-from by statements. Esper window is global. If it is not used with the same stream, the window can hold events of the same type or supertype. When a stream of event arrives, the window function will put the events that match the criteria of the window into a smaller table so that it will not search the entire query to find the matching result.

The syntax of creating named windows is listed below (Esper Team and EsperTech Inc.).

```
create window window_name.view_specifications [as]
        [select list_of_properties from] event_type_or_windowname
        [insert [where filter_expression]]
```

- window_name:
  - Identifier
- view_sepcifications:
  - One or more data window views that define the expiry policy for removing events from the data window.
- select clause and list_of_properties
  - Optional

    - Specify the column names and, implicitly by definition of the event type, the column types of events held by the named window
- Event_type_or_windowname
    - Provides the name of the event type of events held in the data window
- Insert and filter_experession
    - Used if the new named windows is modeled after an existing named window

Another important aspect of Esper temporary window is the window views. Views are applied on windows in order to specify an expiry policy for events from the window, as well as to derive data. Multiple views can be applied to one window.

A window view contains a namespace and a name. There are four types of namespace that follow different names to construct the window view: *win, std, ext,* and *stat*. Namespace *win* and *std* are the most commonly used views and they provide sliding or tumbling data window views. Namespace *ext* is a view that orders events. The *stat* namespace can derive statistical data.

In terms of the project, the time windows involve in many SQL transaction statement. Therefore the time window is our primary concern. One of the most commonly used time window view is *win:time* . It is a moving window extending to the specified time interval into the past based on the system time. Another time window view is *win:time_batch.* This view buffers events and released them every specified time interval in one update. (Esper Team and EsperTech Inc.)

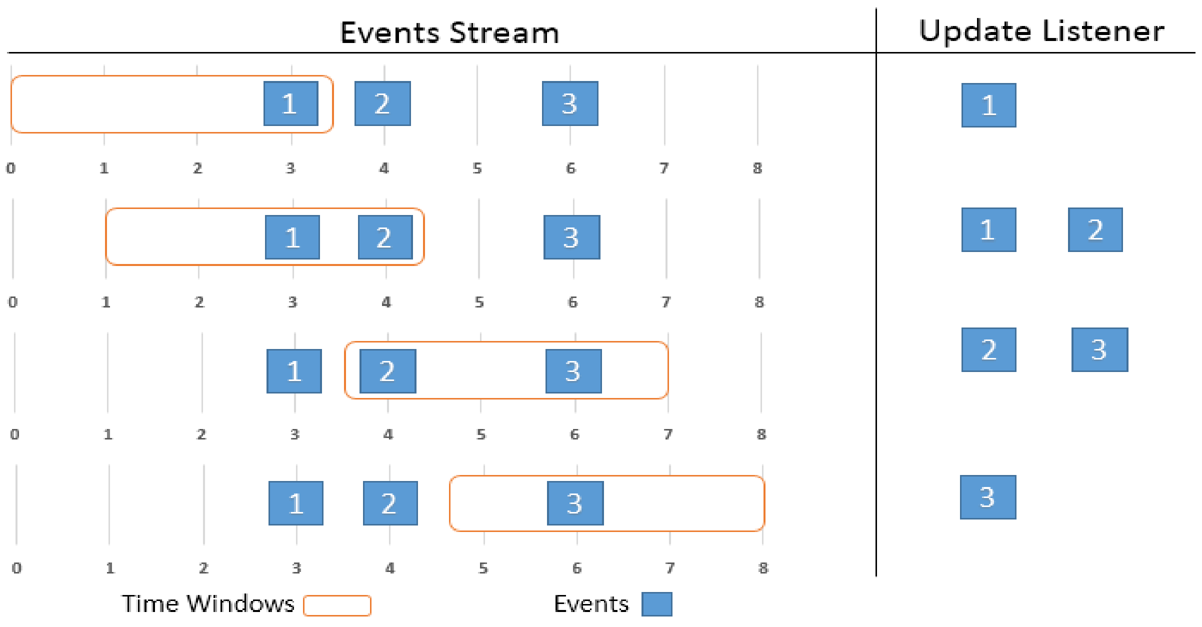


Figure 4.6: SELECT * FROM Withdrawal.win.time(3.5 sec)

The detailed syntax and a description of different window views that we are using are listed below. (Esper Team and EsperTech Inc.)

View	Syntax	Description	Example
Time window	win:time(time period)	Sliding time window extending the specified time interval into the past.	select sum(price) from MyEvent (symbol='GE').win:time(1 sec)
Keep-All window	win:keepall()	The keep-all data window view simply retains all events.	select * from MyEvent.win:keepall()

4.2.1.3 Unique language construct of SQL

Though SQL and Esper are similar, there exist some SQL statements that are not supported by Esper and cannot be translated directly to Esper. The majority of those statements are SQL-specific functions. However, in terms of this project, the SQL functions, given by our sponsor that we are particularly concerned with are *DATEADD*, *DATEDIFF*, *SET*, *DECLARE* and *CONVERT*.

4.2.2 Features and rules

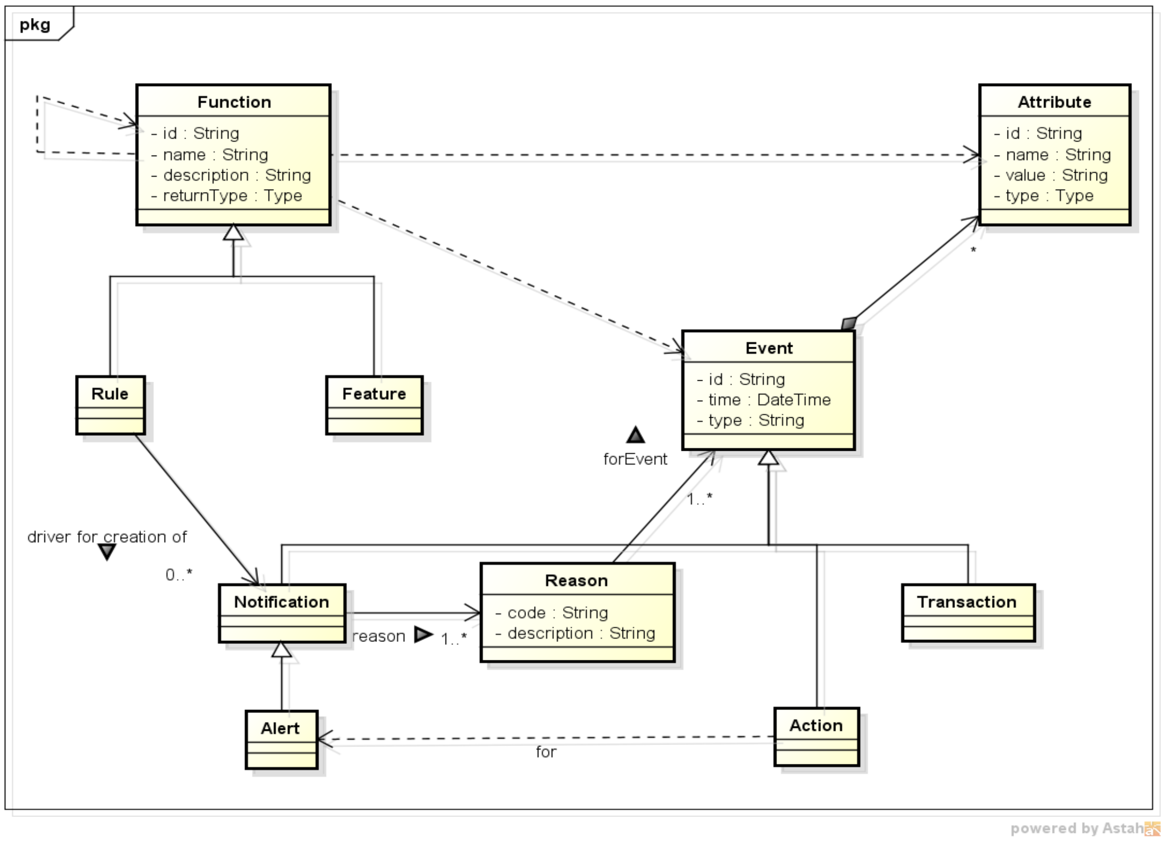


Figure 4.7: Model showing relationship between rules and features

One of the major distinctions of the ACI system is the difference between a feature and a rule. The image above models these differences. Both features and rules are types of functions. A feature is a SQL statement that requires access to a database table and returns a scalar. For example,

```
SELECT COUNT(SD_KEY)
FROM SHORTWINDOW PAST (NOLOCK)
WHERE @SD_TERM_ID = SHORTWINDOW.SD_TERM_ID
AND @DD_DATE < SHORTWINDOW.DD_DATE
AND @DD_DATE > DATEADD(MI, -10, SHORTWINDOW.DD_DATE)
```

Statement 4.6: SQL statement that counts the number of transactions with a certain ID within 10 minutes

This SQL is a feature as it returns a scalar and accesses the SHORTWINDOW, which is a table that holds all transactions within the past three days. A rule is a Boolean expression comprised of constants, scalar expressions, and other rules and features. For example:

```
IF @P1 IN ('00') OR @P2 IN ('01')
BEGIN
        SET @RET = 0
END
```

Statement 4.6: SQL statement that is an example of a rule

This rule is comprised of other rules and features. Statement 4.6 can be broken down into its features and rules as shown below:

```
@P1 IN ('00')
```

Code Fragment 4.4: Feature_01

```
@P2 IN ('01')
```

Code Fragment 4.5: Feature_02

```
IF FEATURE_01 = TRUE or FEATURE_02 = TRUE THEN
        execute action
ENDIF
```

Code Fragment 4.6: Rule 1

This example is composed of two features that are combined into one rule. From these functions, events can be generated. These events and rules can generate notifications, which alert a fraud analyst that a fraudulent transaction has occurred.

Features are very easy to translate to Esper. Statement 4.6 above would simply become in Esper:

```
SELECT COUNT(SD_KEY)
FROM STREAM.win:time(10 minutes)
WHERE TEMP_SD_TERM_ID = SD_TERM_ID
```

Statement 4.7: Statement 4.6 translated to Esper

The *SELECT* and *COUNT* are preserved in the translation. The most important part is the recognition of a window in the *WHERE* clause of the original statement. The fourth and fifth line of Statement 4.7 create a window containing all transactions occurring within 10 minutes of *@DD_DATE*, which is a temporary variable that holds the date of the current transaction. The part of the original statement that compares the *SD_TERM_ID* also stays the same for Esper. The SQL temporary variable, *@SD_TERM_ID*, is converted to *TEMP_SD_TERM_ID*, which is an Esper temporary variable containing the value of the ID being sorted by. The *SD_TERM_ID* of the current transaction is then fetched from the Esper transaction.

Translating rules to Esper is a bit more difficult because ACI requires that rules need to be translated to Java code. Esper's implementation of EPL does not support SQL's Boolean logic of the IF/THEN/ELSE, so we cannot translate the IF/THEN/ELSE directly into Esper as we did for other part. In fact, the way for Esper to handle the Boolean logic of *IF/THEN/ELSE* is through a Java interface *UpdateListener*. The *UpdateListener* has a update function which will contain the IF/ELSE statements. As an example, Statement 4.16 would translate to:

```
public class MyListener implements UpdateListener {

        public void update(EventBean[] newEvents, EventBean[] oldEvents)
        {
                EventBean event = newEvents[0];
                if (event.get("P1").equals('00') || event.get("P2').equals('01')){
                        RET = 0;
                }
        }
}
```

Code Fragment 4.7: Statement 4.6 translated to Esper

Esper's UpdateListeners catch events and evaluate rules against them. In this case, 'P1' and 'P2' are assumed to be contained in an incoming event.

4.2.3 Recognizing and generating time window

As mentioned in the previous chapters, the aspects of windows are native to Esper. As a matter of fact, many fraud detection statements that we were given are time-based and represented by a SQL function that operate on date variables. Yet, Esper does not support those SQL function with the same functionality directly. Therefore, in order to translate the SQL date function into Esper, a time-based window view is the target that we need to recognize and generate.

In the canonical model, date functions in SQL are constructed by *DateOperation* objects. For our translation purpose, we define two types for the *DateOperation*, *DATEADD* and *DATEDIFF* since those are the only SQL date functions in the given examples. For future development, additional SQL functions can be added, as well as customized *DateOperation* types that will utilize more of Esper time-window.

In SQL, the *DATEADD* function returns a specified date with a specified number that is added to an input date. So when it is compared with a date, a time interval will be created. For example, the following SQL statement will create a ten minutes time interval:

```
@DD_DATE > DATEADD(MI, -10, SHORTWINDOW.DD_DATE)
```

Statement 4.8: SQL DATEADD function

According to the example provided by our sponsor, the *DATEADD* functions always occur in a comparison between date variable. Hence, the first type *DateOperation*, *DATEADD*, is translated into Esper directly. Consider the following simple SQL statement:

```
SELECT * FROM SHORTWINDOW
WHERE @DD_DATE < SHORTWINDOW.DD_DATE AND
@DD_DATE > DATEADD(MI, -10, SHORTWINDOW.DD_DATE)
```

Statement 4.9: A Shorter version of Statement 4.6

The essential part of Statement 4.7 is two comparison in the *WHERE* clause. The first one is between *DD_DATE*, and the current time, *SHORTWINDOW.DD_DATE*, and the second one is between *DD_DATE* and the current time minus ten minutes, which is provided by the *DATEADD* function. The two comparisons create a time interval of ten minutes, which is going to be translated into the Esper time window.

The Canonical Model, as represented in Java, of Statement 4.9 would be:

```
        EventSet events = new EventSet("Shortwindow");

        Attribute atdd_date = new Attribute("variable", "@DD_DATE");

        Attribute dd_date = new Attribute("shortwindow", "DD_DATE");

        String[] DATEADDarguments1 = new String[] {"MI", "10",
"SHORTWINDOW.DD_DATE"};

        List<String> DATEADDargslist1 = new ArrayList<String>
(Arrays.asList(DATEADDarguments1));

        SQLFunction DATEADD = new SQLFunction("DATEADD", DATEADDargslist1);

        Comparison comp1 = new Comparison(Comparison.Type.GREATER,
atdd_date,DATEADD);

        Comparison comp2 = new Comparison(Comparison.Type.LESS, atdd_date, dd_date);

        LogicalOperation op1 = new LogicalOperation(LogicalOperation.Type.AND, comp1,
comp2);

        Filter filter = new Filter(op1, events);

        Aggregate all = new Aggregate(Aggregate.Type.ALL, filter, at);
```

Code Fragment 4.8: Java representation of Statement 4.9

Translating the Canonical Model of Statement 4.9 into Esper

```
        SELECT DISTINCT SD_KEY FROM EventSet.win:keepall.win:time(10 min)
```

Statement 4.10: Esper version of Statement 4.9

Another SQL function that will be translated into Esper time window is *DATEDIFF*. *DATEDIFF* will return the count of the difference between the start date and the end date. In our example, DATEDIFF always come with a keyword BETWEEN, and then they construct a time interval. For example, the following SQL statement will create a 72 hours time interval:

```
DATEDIFF(HH, DD_DATE,@DATE) BETWEEN 0 and 72
```

Statement 4.11: DATEDIFF example with BETWEEN

Translating DATEDIFF is more complicated than DATEADD. The way we represent the keyword *BETWEEN* in the canonical model is by a *LogicalOperation AND* and two *Comparisons*. The following will represent the above example:

```
DATEDIFF(HH, DD_DATE,@DATE) > 0
AND
DATEDIFF(HH, DD_DATE,@DATE) < 72
```

Statement 4.12: DATEDIFF example without *BETWEEN*

Since the keyword, *BETWEEN*, is broken into an *AND* in the canonical model, we need to recognize and generate the time-window view in the LogicalOperation. Thus, the separation of a *BETWEEN* causes two conditions in the canonical model that we need to recognize, and generate the time-window view accordingly.

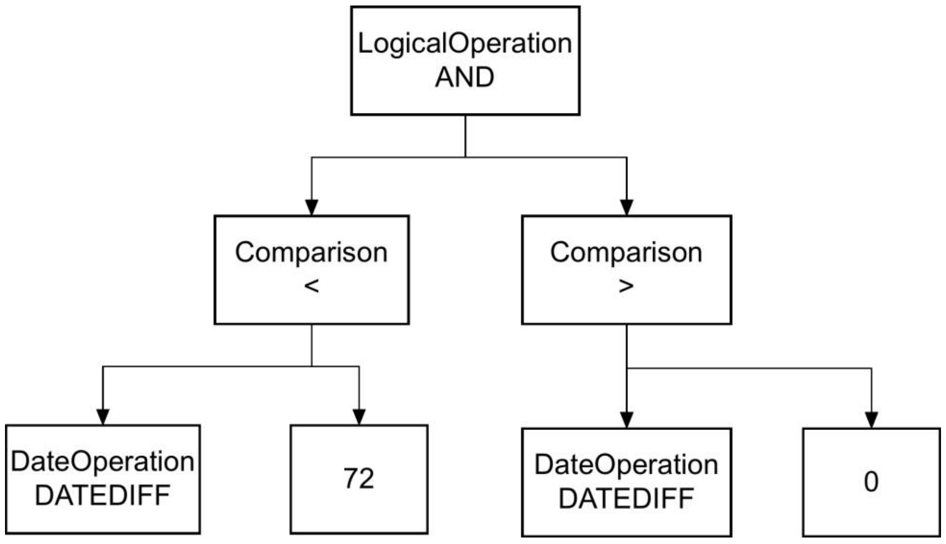


Figure 4.8: Condition 1

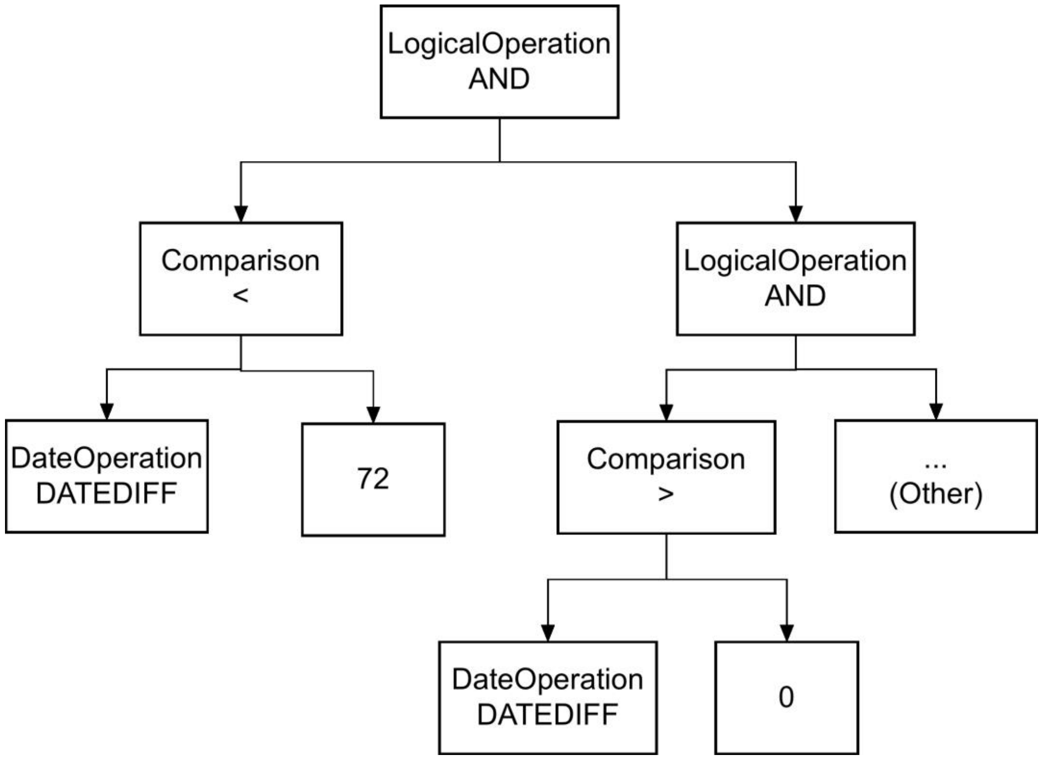


Figure 4.9: Condition 2

Now consider the following SQL statement:

```
SELECT * FROM table1
WHERE DD_DATE < @DATE
AND DATEDIFF(HH, DD_DATE,@DATE) between 0 and 72
```

Statement 4.13: SQL Example of DATEDIFF

Translating the Canonical Model of Statement 4.13 into Esper:

```
SELECT * FROM EventSet.win:time(72 hours)
```

Code Fragment 4.9: Esper version of Statement 4.13

4.2.4 Handling subselect in *FROM* clause

Esper does not fully support most subqueries that SQL provides, especially the subselect in the *FROM* clause.

Consider the following SQL statement, where SELECT is called twice to run through "Table":

```
SELECT * FROM (SELECT TOP 1 * FROM Table WHERE ID = 0)
WHERE State=1
```

Statement 4.14: SQL query with a subquery in a FROM clause

Translating the above statement to Esper:

```
SELECT * FROM (SELECT * FROM Table:std.firstevent() WHERE ID = 0)
WHERE State=1
```

Statement 4.15: Esper version of Statement 4.14

The Esper statement above would not work because Esper has to process the event stream twice, while it can only process an event once. A correct Esper statement should be:

```
SELECT a.col1 FROM Table(ID = 0):std.firstevent() WHERE State = 1
```

Statement 4.16: a correct way to translate the SQL example above into Esper

Another way to convert to the correct Esper syntax is splitting the statement into two streams of events.

```
CREATE WINDOW Win1.win:keepall() AS (SELECT * FROM Table1);
INSERT INTO Win1 SELECT * FROM Table where ID = 0;
SELECT * FROM Win1 WHERE State = 1;
```

Statement 4.17: An alternative, equivalent to Statement 4.16

There is no concrete documentation that points out which cases of complex SQL statements that Esper can handle. Here are several cases that Esper would be able to process complex queries:

- Bounded streams: sub-select the same event using time window

```
SELECT * FROM Account
WHERE balance > (
        SELECT MAX(balance) FROM Account(name = ‘John’).std:lastevent()
)
```

Statement 4.18: Example of a bounded stream

- Sub-select the same event stream within a range

```
SELECT (SELECT * FROM MarketData.std:lastevent()) AS md FROM pattern
[every timer:interval(10 sec)]
```

Statement 4.19: Example of a sub-select

- Select from two different events, processing through each event once

```
        SELECT * FROM Event1, (a IN (SELECT b FROM Event2 AS d WHERE a.a_id =
d.d_id)) AS a
```

Statement 4.20: Example of a subquery in an IN clause

4.2.5 Generating Esper

The object *EsperGenerator* is an object that implements a visitor pattern to generate the EPL statement. It Each object in the Canonical Model has a accept method to accept the visitor. Once a visitor is created and accepted by the object structure, it will get passed through each element, and finally it print out the Esper statement.

4.2.5.1 Accepting the visitor

This example shows how the *EsperGenerator* is created and accepted by the object structure.

```
EsperGenerator esper = new EsperGenerator();
select.accept(esper);
esper.Print();
```

Code Fragment 4.10: Using *EsperGenerator*

4.2.5.2 Nested queries

Looking back at Statement 4.17, splitting a single nested query into three simple queries change the semantic of the query itself. Therefore, the method from Statement 4.16 is chosen so that the Esper query would preserve the same semantic as the SQL query.

```
SELECT col1 FROM
        (SELECT * FROM table1 WHERE col2 < 1000) WHERE eventID > 1
```

Statement 4.21: SQL query with a subquery in the *FROM* clause

```
SELECT col1 FROM table1(* WHERE col2 < 1000) WHERE eventID > 1
```

Statement 4.22: Esper query which is semantically equivalent to SQL query

4.2.6 Gererating Java

Previously, we defined that a rule in the Canonical Model is the SQL statement that has the form of "IF… ELSE". The rule object will be translate into Java, then the Java statement will be combined with the EPL statement to represent the integrated SQL statement.

Similar to the *EsperGenerator*, with the design of the visitor pattern, the object *javaGenerator* gets passed through each element of object structure of the Canonical Model of a rule and then generate the Java Statement.

Consider the Following SQL Statement:

```
IF (SELECT COUNT(*) FROM TABLE) > 1 THEN
        PRINT 'Okay'
ELSE
        PRINT 'FALSE'
```

Statement 4.23: SQL statement of a feature within a rule

The above statement is a simple but typical combination of a rule and a feature. When parsing the SQL statement, the output will contain a Java class *MyListener*, which implements the *UpdateListener* class. The feature, *SELECT COUNT(*) FROM TABLE*, will be separated from the Boolean evaluation to be added to the Esper engine.

The following is the Canonical Model of the Statement 4.23:

```
        CreateAlert iftrue = new CreateAlert(CreateAlert.AlertType.PRINT, "Okay",
"Event1");

        CreateAlert iffalse = new CreateAlert(CreateAlert.AlertType.PRINT, "False",
"Event1");

        RawInteger int_1 = new RawInteger(1);

        EventSet events = new EventSet("Shortwindow");

        Aggregate count = new Aggregate(Aggregate.Type.COUNT, events, new
RawString("*"));

        Filter filter = new Filter(null, events);

        ArrayList<Scalar> selectlist = new ArrayList<Scalar>();

        selectlist.add(count);

        Scalar select = new GetSingleCell(new Select("Shortwindow", selectlist, filter));

        Comparison comp1 = new Comparison(Comparison.Type.GREATER,select,int_1);

        Rule rule = new Rule(iftrue,iffalse,comp1);
```

Code Fragment 4.11: Canonical Model represented in Java of Statement 4.23

```
        EPServiceProvider epService = EPServiceProviderManager.getDefaultProvider();

        String EsperStatement = "SELECT COUNT(*) FROM Shortwindow.win:keepall() ";

        EPStatement FEATURE =
        epService.getEPAdministrator().createEPL(EsperStatement);

        public class MyListener implements UpdateListener {

                public void update(EventBean[] newEvents, EventBean[] oldEvents) {

                        if((newEvents[0].get("COUNT(*)") > 1)){

                                System.out.println("Okay");

                        }

                        else{

                                System.out.println("False");

                        }

                }

        }

FEATURE.addListener(new MyListener());
```

Code Fragment 4.12: Output of parsing Code Fragment 4.11

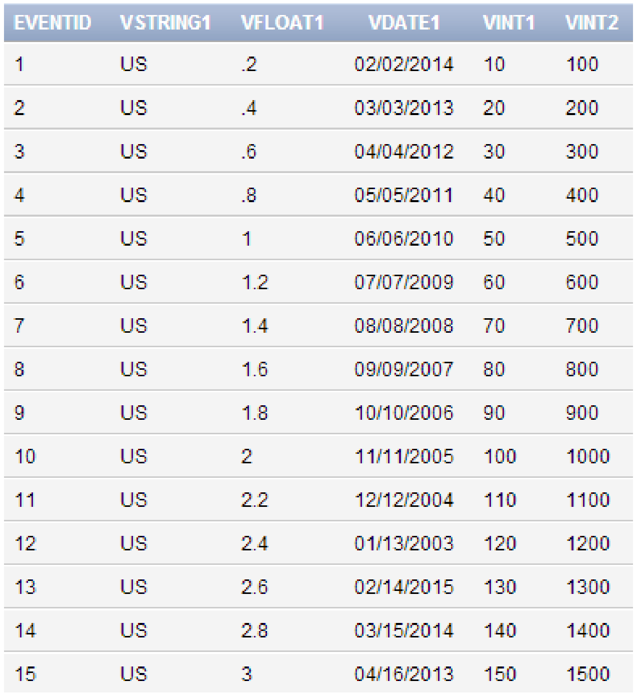
Besides the direct translation, as suggested in our examples, one or more features may be nested in one rule. Yet, *MyListener* class can only handle stream of events. Thus, the *javaGenerator* will combine the features that have the same *EventSet* and generate one integrated Esper statement. Details of those tests can be found in Appendix IV.

5. EVALUATION

5.1 TESTING

5.1.1 Testing Database

For the purpose of testing our program, we built a test database using ACI's Oracle XE database system. The database contains one hundred rows of data and six columns representing different primitive types (INT, FLOAT, STRING, and DATE). In addition, we generated random data to fill the testing database with. The details of the database are shown below.



EVENTID	VSTRING1	VFLOAT1	VDATE1	VINT1	VINT2
1	US	.2	02/02/2014	10	100
2	US	.4	03/03/2013	20	200
3	US	.6	04/04/2012	30	300
4	US	.8	05/05/2011	40	400
5	US	1	06/06/2010	50	500
6	US	1.2	07/07/2009	60	600
7	US	1.4	08/08/2008	70	700
8	US	1.6	09/09/2007	80	800
9	US	1.8	10/10/2006	90	900
10	US	2	11/11/2005	100	1000
11	US	2.2	12/12/2004	110	1100
12	US	2.4	01/13/2003	120	1200
13	US	2.6	02/14/2015	130	1300
14	US	2.8	03/15/2014	140	1400
15	US	3	04/16/2013	150	1500

Figure 5.1: Database test table

Besides the testing database, we also created testing queries that operate on our database so that we can exercise the SQL language. The JUnit tests ensure that first the Esper code that is generated matches the expected output. The Esper query generated in the test is then tested for correctness for syntax and for output against the test database.

5.1.2 Canonical Model to Esper Testing

In order to tests the Esper statement generation, we manually write and construct SQL statements in the form of Canonical Model and run the objects thought the Esper generator. This approach not only helps us with monitoring the Esper Statement output but also provides a clear explanation of how the SQL statements are represented in the form of Canonical Model. A brief introduction of the test cases is the following, for details of all the test cases are in Appendix II:

A. Testing1: select distinct clause, aggregate, from clause, where clause, logical operation, comparison, DATEADD
B. Testing2: select clause, aggregate, from clause, where clause, logical operation, comparison, DATEDIFF
C. Testing3: select clause, aggregate, from clause, where clause, logical operation, comparison, DATEDIFF, inner join
D. Testing4: select clause, aggregate, from clause, where clause, logical operation, comparison, cross join
E. Testing5: select clause, aggregate, from clause, where clause, logical operation, comparison, group by
F. Testing6: select clause, aggregate, from clause, where clause, logical operation, comparison, order by

5.1.3 Esper Syntax checking tests

Appendix III contains the test cases that check the syntax of the Esper statements from the output of Testing2, Testing3, Tesing5, Tesing6, and Testing7. We create streams of random events at run time and send the event through the Esper Statements.

A. TestTesting2: check the syntax of Testing2's Esper transalation result which operates on a single event stream
B. TestTesting3: check the syntax of TestTesting3's Espertranslation result which operates on two event streams.

5.1.4 Comparing SQL and Esper result

To ensure that the results from the auto-generated Esper statements match the original SQL statement, we use the oracle database to test the SQL statements and Esper statement. Yet, due to the lack of a necessary Java library, we do not acquire the necessary environment to run the Esper statement through the SQL table. So our approach is to create streams of events at run time that matches the same table in the Oracle database and send those events through the Esper statement.

The automated test framework captures data from SQL database and convert to a local Esper stream of events; convert SQL query into Esper through the Canonical Model. Then run SQL query and Esper query against the SQL database and Esper database respectively. And finally compare the results of SQL query and Esper query.

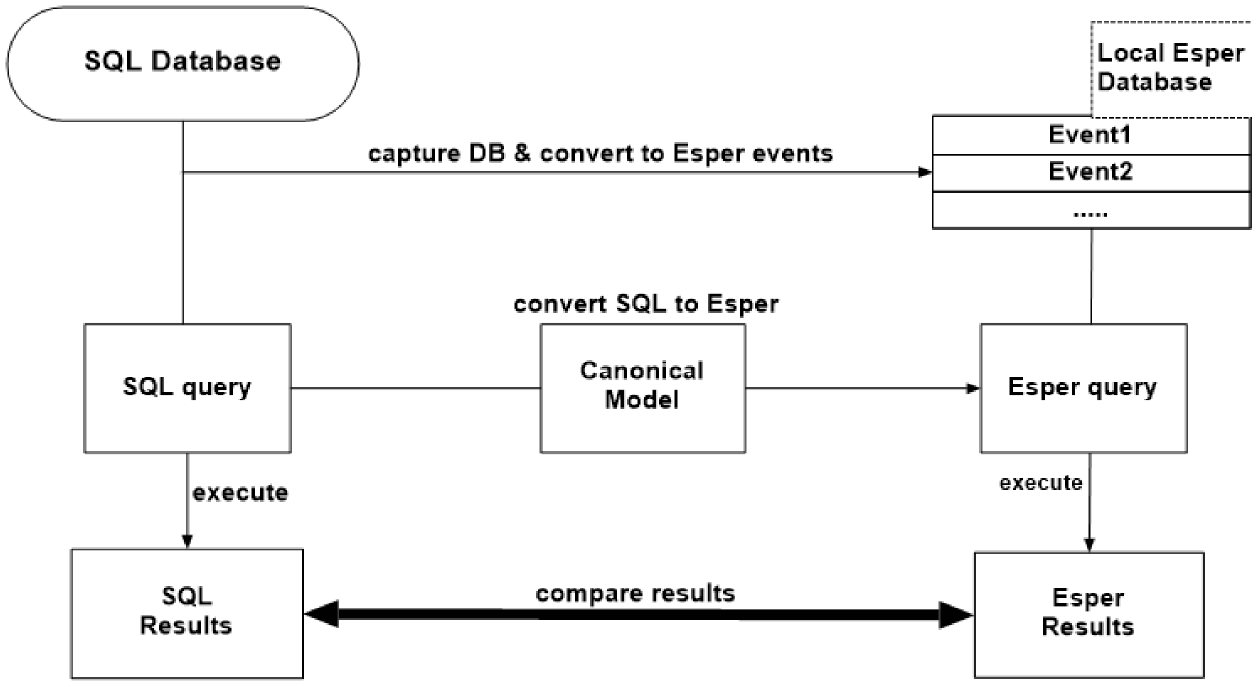


Figure 5.2: Details of the automated test environment

Appendix VII contains tests that demonstrate the test framework above:

A. EventSetup: contains functions that capture data from SQL database and convert to Esper stream of events, convert SQL query into Esper, and compare the results of SQL query and Esper query.

B. TestTesting5: run SQL query and Esper query from Testing5 against the SQL and Esper database respectively, then compare their results. The same goes with TestTesting6 and TestTesting7.

5.1.5 javaGenerator Tests

As mentioned in the previous chapter, Appendix IV contains the tests for javaGenerator that translate from the Canonical Model to JAVA.

A. TestRule3: simple SQL “IF” statement test
B. TestRule4: simple SQL “IF” statement with variable test
C. TestRule5: “IF” statement with two features nested within a comparison
D. TestRule6: one “IF” statement nested to another “IF” statement. Each “IF” statement contains two features nested
E. TestRule7: “IF” statement with a feature that contains *DateOperation DATEADD*

6 Conclusions and future work

At the end of this project, the team reached the conclusion that the canonical model in its current form sufficiently satisfied the needs of SQL and allowed for the expression of both SQL, Esper, and Java. The parsing of the SQL into the model and then into Esper and Java created a satisfactory output that was compilable in Esper. Testing for the code base was very good. The coverage was well beyond 85% and the testing framework is extendable to other languages if it is necessary. However, there is still more work that could be done.

As the project stands, enough SQL and Esper specific constructs and operators are supported for general use were parsed, but there are more constructs and operators than there was time. MSSQL alone has hundreds of MSSQL-specific operations that require special parsing from XML and special parsing to Esper. These operations are used to handle and manipulate variables and raw values. However, without access to a large enough pool of actual RMS sample queries, it was difficult to narrow down the list of features that needed to be supported. From the limited set of queries given, a list of necessary opertions and possible permutations was developed in order to develop a program and model that satisfied the requirements that were known.

Esper, also, has many Esper-specific constructs and operators. Esper includes many syntaxes for different forms of windows and pattern matching. If time had permitted, it could have been useful to implement some of the other window types or try to derive the intent of a SQL query looking for a specific pattern or window. This would greatly improve the efficiency of the outputted Esper and Java code.

Also, for Esper, it would have been interesting to handle cross-stream UpdateListeners. This would have been particularly useful for rules that featured features that operate on separate event streams.

Overall, the project was successful in developing and testing a canonical model for representing different databasing languages specifically with SQL and Esper in mind. However, the simplicity of the model allows for its extension to other languages with relative ease. From the model, it could be possible to design a graphical user interface for the design of fraud detection queries by fraud analysts. The model is so successful a tool that ACI Worldwide is pursuing a patent on it. The SQL to Esper parsing program served to reinforce the model's validity as an essential part of ACI Worldwide's next generation of products and serve as the starting point for the migration of customers to the next generation of products. The next generation of products will be able to enhance the speed and the accuracy of the fraud detection methods and queries by using the canonical model tested in this project as a basis.

Finally, the project was such a success that ACI Worldwide is pursing a follow-up Major Qualifying Project with WPI next year. The project will focus on the use of Storm and other languages for the next generation architecture and will be using the canonical model developed by this project.

7 List of abbreviations

- CEP: Complex Event Processing
- EPL: Event Processing Language
- ESP: Event Stream Processing
- FFD: Financial Fraud Detection
- GSP: General SQL Parser
- RMS: Risk Management System
- SQL: Structured Query Language

8 BIBLIOGRAPHY

Esper Team and EsperTech Inc. *Esper Reference Version 4.10.0*. Web. 6 Jan. 2014. <http://esper.codehaus.org/esper-4.10.0/doc/reference/en-US/pdf/esper_reference.pdf>.

Gayler, Ross, Kate Smith, Vincent Lee, and Phua Clifton . *A Comprehensive Survey of Data Mining-based Fraud Detection Researc*. 2010. Web. 6 Jan. 2014. <http://arxiv.org/abs/1009.6119v1>.

Syeda M, Zhang Y-Q, Pan Y. *Parallel Granular Neural Network for Fast Credit Card Fraud Detection*. 2002. 0-7803-8/02, J IEEE: 572-577

9 APPENDIX

9.1 APPENDIX I: CLASS DICTIONARY OF CANONICAL MODEL

Class: Function

The Function class is designed to be the "handle" of the Canonical model. This means that by accessing a function instance, you have access to the entire model, and can proceed to translate it.

Methods

Method Name	**Signature**	**Description**

Function	Function(String id, String name, String description)
	Function()
		This is the constructor of a function, which initialize the properties if they are passed in.

setID	Public void setID(String id)	Sets the ID of the function
getID	Public String getID()	Returns the ID of the function
setName	Public void setName(String name)	Sets the name of the function
getName	Public String getName()	Returns the name of the function
setDescription	Public void setDescription(String description)	Sets the description of the function
getDescription	Public String getDescription()	Returns the description of the function

Properties

Property Name	**Type**	**Description**
Id	String	An identification string
Name	String	A string used to name the function
Description	String	A string used to describe the intent of the function

Interface: Feature

Every object in the model implements one of two interfaces, Feature and Rule. It is designed to require the implementation of the visitor pattern for translation. The definition of a feature is based off of the ACI definition of a feature in their current RMS product.

Methods

Method Name	**Signature**	**Description**

accept	Public void accept(Visitor visitor)	The method is only a signature designed to implement the visitor method.

Class: EventSet

The EventSet class specifies a set of events. This is used to represent the returned tuples in a select statement, or entire tables.

Methods

Method Name	**Signature**	**Description**
EventSet	Public EventSentProvider(String events)	This is the constructor of an EventSet, and takes a string representing the events
toString	Public String toString()	Returns a string describing the EventSet
accept	Public void accept(Visitor visitor)	Takes a visitor object, and forces it to 'visit' the EventSet

Properties

Property Name	**Type**	**Description**
eventSet	String	A string describing the set of events

Class: Event

The class event is used to describe a specific tuple, which will be necessary in certain SQL statements

Methods

Method Name	**Signature**	**Description**
Event	Public Event(String id, Date timestamp, String source, String type)	This is the constructor of an Event. It initializes the events properties
Accept	Public void accept(Visitor visitor)	Takes a visitor object, and forces it to 'visit' the Event

Properties

Property Name	**Type**	**Description**
Id	String	An identification string

timeStamp	Date	A date identifying the creation time of the tuple represented by the Event
source	String	A string identifying the source of the tuple represented by the Event
type	String	A string identifying the data type of the tuple represented by the Event

Class: SetOperation

The SetOperation class is used to represent any setOperations from the original SQL statement, such as join, natural join, left full join, etc. It extends EventSet, and therefore all of its methods.

Methods

Method Name	**Signature**	**Description**
SetOperation	SetOperation(EventSet left, EventSet right, Filter filter, Type type)	This is the constructor of a SetOperation.
setOn	Public void setOn(Filter filt)	Sets the filter
setLeft	Public void setLeft(EventSet es)	Sets the left property as an EventSet
setRight	Public void setRight(EventSet es)	Sets the right property as an EventSet

Properties

Property Name	**Type**	**Description**
Left, Right	EventSet	The two EventSets which are being represented in the setOperation
onStatement	Filter	Represents the “on” statement of an SQL query
Type	Enum Type{ UNION, INTERSECTION, DIFFERENCE, INNERJOIN, NATURALJOIN, EQUIJOIN, LEFTOUTERJOIN, RIGHTOUTERJOIN, FULLOUTERJOIN, CROSSJOIN}	An Enum used to describe the type of setOperation

Class: Group

The Group class represents SQL grouping. It takes a list of Attributes with which it performs the group operation. Group extends the SetOperation class.

Methods

Method Name	**Signature**	**Description**
Group	Group(EventSet eventSet, ArrayList<Attribute> groupOn)	This is the constructor of a function, which initialize the properties if they are passed in.

Properties

Property Name	**Type**	**Description**
eventSet	EventSet	The EventSet which is being manipulated
groupOn	ArrayList<Attribute>	A list of the fields which are being grouped

Class: Limit

The Limit class represents a limit in SQL queries. This is also used to implement similarly used constraints, such as TOP. Limit extends the SetOperation class.

Methods

Method Name	**Signature**	**Description**
Limit	Limit(EventSet eventSet, int start, int end)	This is the constructor for the Limit class

Properties

Property Name	**Type**	**Description**
eventSet	EventSet	The EventSet which is being manipulated
start	int	Starting location of the limit

end	int	Ending location of the limit

Class: Order

The Order class represents an ordering operation on an EventSet. This class extends the SetOperation class.

Methods

Method Name	**Signature**	**Description**
Order	Order(EventSet eventSet, Attribute orderOn, Direction direction)	This is the constructor for the Order class

Properties

Property Name	**Type**	**Description**
eventSet	EventSet	The EventSet which is being manipulated
orderOn	Attribute	Attribute to order the set by
Direction	Enum Direction	Enum of the valid directions of ordering (ASC, DESC)
direction	Direction	Direction of the order operation

Class: Filter

The Filter class is used to represent constraints and restrictions on an EventSet. An example would be an SQL “where” clause. Filter extends EventSet.

Methods

Method Name	**Signature**	**Description**
Filter	Filter(Prediicate pred, EventSet set)	This is the constructor of a filter. It initializes the properties
setID	Public void setID(String id)	Sets the ID of the function
getID	Public String getID()	Returns the ID of the function

setName	Public void setName(String name)	Sets the name of the function
getName	Public String getName()	Returns the name of the function
setDescription	Public void setDescription(String description)	Sets the description of the function
getDescription	Public String getDescription()	Returns the description of the function

Properties

Property Name	**Type**	**Description**
Id	String	An identification string
Name	String	A string used to name the function
Description	String	A string used to describe the intent of the function

Class: Aggregate

The Aggregate class is used to represent aggregate functions such as COUNT, SUM, AVG, etc. It has a type, which indicate which aggregate function it is representing, and an EventSet and Attribute that the function operates on.

Methods

Method Name	**Signature**	**Description**
Aggregate	Aggregate(Type type, EventSet set, Attribute at) Aggregate()	This is the constructor for Aggregate. It initializes the properties of the instance if they are passed.
getType	Public String getType()	Returns a string representation of the type
getSet	Public EventSet getSet()	Returns the EventSet
getAt	Public Attribute getAt()	Returns the Attribute
setType	Public void setType(Type type)	Sets the type of the Aggregate
setSet	Public void setSet(EventSet set)	Sets the EventSet
setAt	Public void setAt(Attribute at)	Sets the Attribute
accept	Public void accept(Visitor visitor)	Accepts the visitor, and calls visitor.visit(this). It then passes the visitor to the EventSet

Properties

Property	**Type**	**Description**

Name		
type	enum Type	The type of aggregate function being applied
attrib	Scalar	The scalar value which is being aggregated
eventSet	EventSet	The EventSet the attribute is from

Class: Select

The Select class is used to represent a select statement in SQL.

Methods

Method Name	Signature	Description
accept	Accept(Visitor visitor)	Forces all Scalars to have the accept method to implement the visitor design patern

Properties

Property Name	Type	Description
scalarList	ArrayList<Scalar>	List of Scalars which will be selected
eventSet	EventSet	The table which the Select operates on
isDistinct	Boolean	Expresses whether the selection is DISTINCT

Interface: Scalar

The Scalar interface is used as an identifier for objects that represent parts of an SQL statement that return a scalar value, such as logical or comparison operations, or even aggregates.

Methods

Method Name	Signature	Description
accept	Accept(Visitor visitor)	Forces all Scalars to have the accept method to implement the visitor design patern

Class: Attribute

The Attribute class is used to represent a field in an SQL table.

Methods

Method Name	Signature	Description
Attribute	Attribute(String id, String value)	This is the constructor for Attribute. It initializes the properties of the instance if

	Attribute(String value)	they are passed.
setTable	Public void setTable(String table)	Sets the table of the Attribute
setColumn	Public void setColumn(String column)	Sets the column of the Attribute
getTable	Public String getTable()	Returns the Table
getColumn	Public void getColumn()	Returns the Column

Properties

Property Name	**Type**	**Description**
value	String	String representation of the SQL field
id	String	The ID of the Attribute
isDistinct	Boolean	Expresses whether the Attribute is labeled DISTINCT or not

Class: AttributeReference

The AttributeReference class references an Attribute. It is used to get a specific instance of the Attribute.

Methods

Method Name	**Signature**	**Description**
accept	Public void accept()	Forces all Scalars to have the accept method to implement the visitor design patern

Properties

Property Name	**Type**	**Description**
attrName	String	Name of the Attribute being referenced

Class: GetSingleCell

The GetSingleCell class is used to assert that a table returns a single value. As an example, it is used in comparisons between singular values and subqueries.

Methods

Method Name	**Signature**	**Description**
Accept	Public accept(Visitor visitor)	Forces the visitor to visit the left predicate, then this, then the right predicate

Properties

Property Name	**Type**	**Description**
set	Select	The Select object which is forced into a single value

Interface: Predicate

The Predicate interface is used to identify any model object that represents a part of an SQL statement that would return a Boolean true or false value. It implements Scalar and its accept method exactly.

Abstract Class: UnaryOperation

The UnaryOperation abstract class represents functionality which performs some operation on a single operation.

Methods

Method Name	**Signature**	**Description**
Accept	Public accept(Visitor visitor)	Forces the visitor to visit the left predicate, then this, then the right predicate

Properties

Property Name	**Type**	**Description**
operand	Scalar	The operand which the UnaryOperation operates on

Class: Case

The Case class is used to represent SQL case statements. It is effectively a list comprised of a smaller class, called CaseInstance, which contains a clause for each possible case. Case extends UnaryOperation.

Methods

Method Name	**Signature**	**Description**
Case	Case(Scalar operand, ArrayList<CaseInstance> cases)	The constructor for a Case object.
Accept	Public accept(Visitor visitor)	Forces the visitor to visit the left predicate, then this, then the right predicate

Properties

Property Name	**Type**	**Description**

operand	Scalar	The operand which the UnaryOperation operates on
cases	ArrayList<CaseInstance>	List of cases to check for on the operand.

Class: StringOperation

This class handles operations on string objects, such as taking a substring. StringOperation extends UnaryOperation.

Methods

Method Name	**Signature**	**Description**
StringOperation	StringOperation(Predicate operand)	This is the constructor for StringOperation.
Accept	public void accept(Visitor visitor)	Forces the visitor to visit this object

Properties

Property Name	**Type**	**Description**
type	Enum UnaryOperatorType	The type of operation to be performed on the predicate
operand	Predicate	Represents the predicate the NOT statement operates on

Class: Variable

The Variable class represents a stored variable in SQL. It has a name, type and value.

Methods

Method Name	**Signature**	**Description**
Variable	Variable(String name, Type type, Value value)	This is the constructor for a Variable object

Properties

Property Name	**Type**	**Description**

name	String	The name of the variable as listed in SQL
type	Enum Type	The type of variable (ex. int, varchar)
value	String	String representation of the value of the variable

Class: DateOperation

The DateOperation class represents any operation performed specifically on date type objects in SQL. This is particularly of interest for identifying time windows.

Methods

Method Name	**Signature**	**Description**
DateOperation	DateOperation(Name name, List<Scalar> argslist)	This is the constructor for DateOperation.
Accept	public void accept()	Forces the visitor to visit this object

Properties

Property Name	**Type**	**Description**
Name	Enum Name	A Enum which lists the valid names of date operations
name	Name	The name of the function in SQL
listOfArgs	List<Scalar>	List of the parameters which the function takes. The size may vary depending on the function.

Class: LogicalOperation

The LogicalOperation class is used to represent any AND, OR, or XOR statements in SQL. It contains a left and right operator of type Predicate, and implements predicate itself.

Methods

Method Name	**Signature**	**Description**
LogicalOperation	LogicalOperation(Type type, Predicate left, Predicate right)	This is the constructor for LogicalOperation. It initializes the properties of the instance if they are passed.

getType	Public String getType()	Returns a string representation of the type
getLeft	Public Predicate getLeft()	Returns the left Predicate property
getRight	Public Predicate getRight()	Returns the right Predicate property
Accept	Public accept(Visitor visitor)	Forces the visitor to visit the left predicate, then this, then the right predicate

Properties

Property Name	**Type**	**Description**
operation	Enum ArithmeticOperationType	Represents the logical operation to be performed
left, right	Predicate	The left and right predicates the logical operator operates on

Class: UnaryLogicalOperation

The UnaryLogicalOperation class is used to represent a negative on a logicalOperation, or a NOT statement in SQL. It implements the Predicate interface

Methods

Method Name	**Signature**	**Description**
UnaryLogicalOperation	UnaryLogicalOperation(Predicate operated)	This is the constructor for UnaryLogicalOperation. It initializes the properties of the instance if they are passed.

Properties

Property Name	**Type**	**Description**
type	Enum UnaryOperatorType	The type of operation to be performed on the predicate
operand	Predicate	Represents the predicate the NOT statement operates on

Class: Comparison

The Comparison class is used to represent scalar comparisons in the SQL statement such as <, >, =, etc. The class implements the Predicate interface.

Methods

Method Name	**Signature**	**Description**
Comparison	Comparison(Type type, Scalar left, Scalar right)	This is the constructor for Comparison. It initializes the properties of the instance if they are passed.
getType	Public String getType()	Returns a string representation of the type
getRight	Public Scalar getRight()	Returns the right property of the Comparison
getLeft	Public Scalar getLeft()	Returns the left property of the comparison

Properties

Property Name	**Type**	**Description**
Type	Enum Type	Represents the aggregate function the instance represents
Left, right	Scalar	The left and right Scalars to be compared

Class: ArithmeticOperation

The ArithmeticOperation class is used to represent simple arithmetic functions in the SQL statement such as +, -, *, etc.

Methods

Method Name	**Signature**	**Description**
ArithmeticOperation	ArithmeticOperation(Type type, Scalar left, Scalar right)	This is the constructor for ArithmeticOperation. It initializes the properties of the instance if they are passed.
getType	Public String getType()	Returns a string representation of the type
getLeft	Public Scalar getLeft()	Returns the left Scalar property
getRight	Public Scalar getRight()	Returns the right Scalar property
Accept	Public accept(Visitor visitor)	Forces the visitor to visit the left predicate, then this, then the right predicate

Properties

Property Name	**Type**	**Description**
Type	Enum ArithmeticOperationType	Represents the arithmetic operation the instance represents
Left, right	Scalar	The left and right scalars the logical operator operates on

Class: UnaryArithmeticOperation

The UnaryArithmeticOperation class is used to represent arithmetic operations that only operate on one scalar, such as ++, --, sin, cos, etc.

Methods

Method Name	Signature	Description
UnaryArithmeticOperation	UnaryLogicalOperation(Type type, Scalar operated)	This is the constructor for UnaryArithmeticOperation. It initializes the properties of the instance if they are passed.

Properties

Property Name	Type	Description
Operated	Scalar	Represents the Scalar the arithmetic statement operates on
Type	Type	String representation of the type of unary arithmetic function.

Class: EsperGenerator

The EsperGenerator class is a visitor used to parse the model instance. While parsing the instance, the generator modifies a string stored within by adding relevant code for each model object it visits.

Methods

Method Name	Signature	Description
EsperGenerator	EsperGenerator()	This is the constructor for EsperGenerator. It initializes the properties of the instance
visit	Public void visit(*Model Objects*)	Modifies the internal properties to reflect the Esper code necessary to represent the model objects it visits
print	Public String print()	Prints the Esper code to represent the model instance to the screen

Properties

Property Name	Type	Description
esper	String	Internal string that is modified to represent the query
window	String	Internal string to represent the window parameters if the esper translation should require it
esper2	String	Internal string that is modified to represent the query after the window expression

Class: JavaGenerator

The JavaGenerator class is a visitor used to parse Rules. While Features are parsed as Esper, Rules are parsed as Java code.

Methods

Method Name	**Signature**	**Description**
javaGenerator	javaGenerator()	This is the constructor for EsperGenerator. It initializes the properties of the instance
visit	Public void visit(*Model Objects*)	Modifies the internal properties to reflect the Java code necessary to represent the model objects it visits
print	Public String print()	Prints the Java code to represent the model instance to the screen

Properties

Property Name	**Type**	**Description**
java	String	Internal string that is modified to represent the query
window	String	Internal string to represent the window parameters if the esper translation should require it
esper	String	Internal string that represents the Esper code in the Feature portion of the code

9.2 APPENDIX II: TESTING THE TRANSFORMATION OF THE CANONICAL MODEL TO ESPER

Canonical Model Objects: Common canonical model objects that will be used in the test files

```
public class CMObjects {
        public static EventSet event1 = new EventSet("acimqp1");
        public static EventSet event2 = new EventSet("Event2");
        public static ProjectAll all = new ProjectAll(null);

        public static Attribute vEventID = new Attribute("eventID");
        public static Attribute vInt1 = new Attribute("vInt1");
        public static Attribute vInt2 = new Attribute("vInt2");
        public static Attribute vString1 = new Attribute("vString1");
        public static Attribute vFloat = new Attribute("vFloat1");

        public static Attribute e1EventID = new Attribute("acimqp1", "eventID");
        public static Attribute e1Int1 = new Attribute("acimqp1", "vInt1");
        public static Attribute e1Int2 = new Attribute("acimqp1", "vInt2");
        public static Attribute e1String1 = new Attribute("acimqp1", "vString1");
        public static Attribute e1Float = new Attribute("acimqp1", "vFloat1");
```

```
	public static RawString rUS = new RawString("'US'");
	public static RawString rCA = new RawString("'CA'");
	public static RawString rUK = new RawString("'UK'");
	public static RawString rRU = new RawString("'RU'");
}
```

Testing1:

```
public class Testing {

	/*
	 * Original:
	 SELECT COUNT(DISTINCT(SD_KEY))+1
		FROM SHORTWINDOW PAST (NOLOCK)
		WHERE @SD_TERM_ID = SHORTWINDOW.SD_TERM_ID
		AND @DD_DATE < SHORTWINDOW.DD_DATE
		AND @DD_DATE > DATEADD(MI, -10, SHORTWINDOW.DD_DATE)
		AND MD_TRAN_AMT1 >= 100
		AND @SD_KEY <> SHORTWINDOW.SD_KEY
		AND @SD_TERM_CNTRY IN ('XX','XZ')
	 */

	@Test
	public void test() {
		String expected = "SELECT DISTINCT COUNT(SD_KEY) FROM 
Shortwindow.win:keepall().win:time(10 min)"
				+ "\n\tWHERE SD_Term_Country = 'XX'"
				+ "\n\tOR SD_Term_Country = 'XZ'"
				+ "\n\tAND Shortwindow.SD_Term_ID = SD_Term_ID"
				+ "\n\tAND DD_DATE = Shortwindow.DD_DATE"
				+ "\n\tAND SD_KEY <> Shortwindow.SD_KEY"
				+ "\n\tAND MD_Tran_Amt1 >= 100";
		assertEquals(genEsper().getEsper(), expected);
	}

	@SuppressWarnings("unused")
	public static EsperGenerator genEsper() {
		Variable at_SD_Term_Country = new 
Variable("SD_Term_ID",Variable.Type.INT,"1");
		Variable at_DD_DATE = new 
Variable("DD_DATE",Variable.Type.VARCHAR,"2010.1.1");

		EventSet events = new EventSet("Shortwindow");
		Attribute sdkey = new Attribute("Shortwindow", "SD_KEY");
```

```
                    Variable Vatdd_date = new Variable("DD_DATE");

                    Attribute country = new Attribute("variable", "SD_Term_Country");
                    Attribute sd_key = new Attribute("SD_KEY");
                    Attribute sd_term_id = new Attribute("Shortwindow", "SD_Term_ID");
                    Attribute atsd_term_id = new Attribute("variable", "SD_Term_ID");

                    Attribute dd_date = new Attribute("Shortwindow", "DD_DATE");

                    Attribute md_tran_amt1 = new Attribute("variable", "MD_Tran_Amt1");
                    Attribute atsd_key = new Attribute("variable", "SD_KEY");
                    RawInteger md_tran_amtcomp = new RawInteger(100);

                    RawString MI = new RawString("MI");
                    RawInteger neg_10 = new RawInteger(-10);
                    RawString SHORTWINDOW_DD_DATE = new
RawString("SHORTWINDOW.DD_DATE");

                    Scalar[] arguments = new Scalar[] {MI, neg_10,
SHORTWINDOW_DD_DATE};
                    List<Scalar> argslist = new ArrayList<Scalar>(Arrays.asList(arguments));

                    DateOperation DATEADD = new
DateOperation(DateOperation.Name.DATEADD,argslist);

                    Comparison comp1 = new Comparison(Comparison.Type.EQUALS, country, new
RawString("'XX'"));
                    Comparison comp2 = new Comparison(Comparison.Type.EQUALS, country,
new RawString("'XZ'"));
                    Comparison comp3 = new Comparison(Comparison.Type.EQUALS, sd_term_id,
atsd_term_id);
                    Comparison comp4 = new Comparison(Comparison.Type.EQUALS, Vatdd_date,
dd_date);
                    Comparison comp5 = new Comparison(Comparison.Type.NOTEQUALS,
atsd_key, sdkey);
                    Comparison comp6 = new Comparison(Comparison.Type.GREATEREQUALS,
md_tran_amt1, md_tran_amtcomp);

                   Comparison comp7 = new Comparison(Comparison.Type.GREATER, Vatdd_date,
DATEADD);

                    LogicalOperation op1 = new LogicalOperation(LogicalOperation.Type.OR,
comp1, comp2);
                    LogicalOperation op2 = new LogicalOperation(LogicalOperation.Type.AND,op1,
comp3);
```

```
		LogicalOperation op3 = new LogicalOperation(LogicalOperation.Type.AND,op2,
comp4);
		LogicalOperation op4 = new LogicalOperation(LogicalOperation.Type.AND,op3,
comp5);
		LogicalOperation op5 = new LogicalOperation(LogicalOperation.Type.AND,op4,
comp6);

		LogicalOperation op6 = new LogicalOperation(LogicalOperation.Type.AND,op5,
comp7);

		Filter filter = new Filter(op6, events);

		Aggregate count = new Aggregate(Aggregate.Type.COUNT, events, sd_key);

		ArrayList<Scalar> distinctlist = new ArrayList<Scalar>();
		distinctlist.add(count);
		Select distinct = new Select("Shortwindow", distinctlist, filter, true);

		EsperGenerator esper = new EsperGenerator();

		distinct.accept(esper);
		return esper;
	}
}
```

Output:

```
SELECT DISTINCT COUNT(SD_KEY) FROM Shortwindow.win:keepall().win:time(10 min)
	WHERE SD_Term_Country = 'XX'
	SD_Term_Country = 'XZ'
	AND Shortwindow.SD_Term_ID = SD_Term_ID
	AND DD_DATE = Shortwindow.DD_DATE
	AND SD_KEY <> Shortwindow.SD_KEY
	AND MD_Tran_Amt1 >= 100
```

Testing2:

```
public class Testing2 {

	/*
	* Original:
	SELECT Count(*)
		FROM SHORTWINDOW
		WHERE MD_TRAN_AMT1 < 1000
		AND SD_Term_Country = 'US'
		AND DATEDIFF(HH,DD_DATE,@DD_DATE) BETWEEN 0 AND 72
	*/

	@Test
	public void test() {
		String expected = "SELECT COUNT(*) FROM
acimqp1.win:keepall().win:time(72 days)"
					+ "\n\tWHERE acimqp1.vInt1 < 1000"
					+ "\n\tAND acimqp1.vString1 = 'US'";
		EsperGenerator esper = genEsper();
		esper.Print();
		assertEquals(expected, esper.getEsper());
	}

	public static EsperGenerator genEsper() {
		RawInteger md_tran_amtcomp = new RawInteger(1000);
		RawInteger date_comp0 = new RawInteger(0);
		RawInteger date_comp72 = new RawInteger(72);

		RawString HH = new RawString("DD");
		RawString DD_DATE = new RawString("DD_DATE");
		RawString atDD_DATE = new RawString("@DD_DATE");

		Scalar[] arguments = new Scalar[] {HH, DD_DATE, atDD_DATE};
		List<Scalar> argslist = new ArrayList<Scalar>(Arrays.asList(arguments));

		DateOperation DATEDIFF = new
DateOperation(DateOperation.Name.DATEDIFF,argslist);

		//"BETWEEN" in SQL is translated into a LogicalOperation "AND" and two
Comparison

		Comparison comp1 = new Comparison(Comparison.Type.LESS,
CMObjects.e1Int1, md_tran_amtcomp);
		Comparison comp2 = new Comparison(Comparison.Type.EQUALS,
CMObjects.e1String1, CMObjects.rUS);
		Comparison comp3 = new Comparison(Comparison.Type.GREATER,
DATEDIFF, date_comp0);
		Comparison comp4 = new Comparison(Comparison.Type.LESS, DATEDIFF,
date_comp72);
```

```
			LogicalOperation op1 = new LogicalOperation(LogicalOperation.Type.AND,
comp1, comp2);
			LogicalOperation op2 = new LogicalOperation(LogicalOperation.Type.AND, op1,
comp3);
			LogicalOperation op3 = new LogicalOperation(LogicalOperation.Type.AND, op2,
comp4);

			Filter filter = new Filter(op3, CMObjects.event1);

			Aggregate count = new Aggregate(Aggregate.Type.COUNT, CMObjects.event1,
CMObjects.all);

			ArrayList<Scalar> selectlist = new ArrayList<Scalar>();
			selectlist.add(count);
			Select select = new Select("Shortwindow", selectlist, filter);

			EsperGenerator esper = new EsperGenerator();
			select.accept(esper);
			return esper;
	}
}
```

Output:

```
SELECT COUNT(*) FROM Shortwindow.win:keepall().win:time(72 days)
	WHERE Shortwindow.MD_Tran_Amt1 < 1000
	AND Shortwindow.SD_Term_Country = 'US'
```

Testing3:

```
public class Testing3 {

/*
 * Original:
 SELECT COUNT(*)
	FROM SHORTWINDOW PAST (NOLOCK)
	WHERE DATEDIFF(DD,DD_DATE,@DD_DATE) BETWEEN 0 AND 12
	AND SD_TERM_CNTRY IN ('US','CA')
	AND MD_TRAN_AMT1 >= 100
 */

@Test
public void test() {
	String expected = "SELECT COUNT(*) FROM Table1.win:keepall() INNER JOIN
```

```
Table2.win:keepall().win:time(12 days) ON Table1.MD_Tran_Amt1 = Table2.MD_Tran_Amt1"
                              + "\n\tWHERE Table1.SD_Term_Country = 'US'"
                              + "\n\tOR Table1.SD_Term_Country = 'CA'";
          assertEquals(genEsper().Print(), expected);
}

public static EsperGenerator genEsper() {

          EventSet table1 = new EventSet("Table1");
          EventSet table2 = new EventSet("Table2");

          SetOperation setOp = new SetOperation("events", table1, table2,SetOperation.Type.INNERJOIN);

          Attribute all = new Attribute("*");
          Attribute country = new Attribute("Table1", "SD_Term_Country");
          Attribute md_tran_amt_t1 = new Attribute("Table1", "MD_Tran_Amt1");
          Attribute md_tran_amt_t2 = new Attribute("Table2", "MD_Tran_Amt1");

          RawInteger date_comp0 = new RawInteger(0);
          RawInteger date_comp12 = new RawInteger(12);
          RawString DD = new RawString("DD");
          RawString DD_DATE = new RawString("DD_DATE");
          RawString atDD_DATE = new RawString("@DD_DATE");
          Scalar[] arguments = new Scalar[] {DD, DD_DATE, atDD_DATE};
          List<Scalar> argslist = new ArrayList<Scalar>(Arrays.asList(arguments));

          DateOperation DATEDIFF = new DateOperation(DateOperation.Name.DATEDIFF,argslist);

          Comparison comp1 = new Comparison(Comparison.Type.GREATER, DATEDIFF, date_comp0);
          Comparison comp2 = new Comparison(Comparison.Type.LESS, DATEDIFF, date_comp12);
          Comparison comp3 = new Comparison(Comparison.Type.EQUALS, country, new
Attribute("constant", "'US'"));
          Comparison comp4 = new Comparison(Comparison.Type.EQUALS, country,  new
Attribute("constant", "'CA'"));
          Comparison comp6 = new Comparison(Comparison.Type.EQUALS, md_tran_amt_t1,
md_tran_amt_t2);

          LogicalOperation op1 = new LogicalOperation(LogicalOperation.Type.AND, comp1, comp2);
          LogicalOperation op2 = new LogicalOperation(LogicalOperation.Type.AND,op1, comp3);
          LogicalOperation op3 = new LogicalOperation(LogicalOperation.Type.OR,op2, comp4);

          Filter filter = new Filter(op3, setOp);
          Filter JoinOn = new Filter(comp6,setOp);
          setOp.setOn(JoinOn);
          Aggregate count = new Aggregate(Aggregate.Type.COUNT, null, all);
```

```
            ArrayList<Scalar> selectlist = new ArrayList<Scalar>();
            selectlist.add(count);
            Select select = new Select(null, selectlist, filter);

            EsperGenerator esper = new EsperGenerator();
            select.accept(esper);
            return esper;
        }
}
```

Output:

```
SELECT COUNT(*)
        FROM Table1.win:keepall() INNER JOIN Table2.win:keepall().win:time(12 days)
        ON Table1.MD_Tran_Amt1 = Table2.MD_Tran_Amt1
        WHERE Table1.SD_Term_Country = 'US'
        OR Table1.SD_Term_Country = 'CA'
```

Testing4:

```
        public class Testing4 {

        @Test
        public void test() {
                String expected = "SELECT * FROM Table1.win:keepall() CROSS JOIN
Table2.win:keepall()  ON vInt = 300"
                                + "\n\tWHERE vInt = 300";
                assertEquals(genEsper().Print(), expected);
        }

        public static EsperGenerator genEsper() {
                EventSet table1 = new EventSet("Table1");
                EventSet table2 = new EventSet("Table2");
                Attribute all = new Attribute ("*");
                SetOperation op1 = new SetOperation("", table1, table2,
SetOperation.Type.CROSSJOIN);
                Attribute int1 = new Attribute("variable", "vInt");
                Attribute vint1 = new Attribute("constant", "300");
                Comparison lop1 = new Comparison(Comparison.Type.EQUALS, int1, vint1);

                Filter filter = new Filter(lop1, op1);
                op1.setOn(filter);

                ArrayList<Scalar> selectlist = new ArrayList<Scalar>();
                selectlist.add(all);
```

```
		Select select = new Select("", selectlist, filter);

		EsperGenerator esper = new EsperGenerator();
		select.accept(esper);
		return esper;
	}
}
```

Output:

```
SELECT * FROM Table1.win:keepall() CROSS JOIN Table2.win:keepall()
	ON vInt = 300 WHERE vInt = 300
```

Testing5:

```
public class Testing5 {

	@Test
	public void test() {
		String expected = "SELECT COUNT(*) FROM acimqp1.win:keepall() "
				+ "\n\tWHERE acimqp1.vInt2 < 1000"
				+ "\n\tAND acimqp1.vFloat1 < 50.0"
				+ "\n\tGROUP BY vInt2";
		EsperGenerator esper = genEsper();
		esper.Print();
		assertEquals(expected, esper.getEsper());
	}

	public static EsperGenerator genEsper() {
		RawInteger md_tran_amtcomp = new RawInteger(1000);
		RawFloat fl_amt = new RawFloat(50.0f);
		Comparison comp1 = new Comparison(Comparison.Type.LESS,
CMObjects.e1Int2, md_tran_amtcomp);
		Comparison comp2 = new Comparison(Comparison.Type.LESS,
CMObjects.e1Float, fl_amt);
		LogicalOperation op1 = new LogicalOperation(LogicalOperation.Type.AND,
comp1, comp2);
		Filter filter = new Filter(op1, CMObjects.event1);
		Attribute groupOn = new Attribute("vInt2");

		EventSet group = new Group(filter,groupOn);
		Aggregate count = new Aggregate(Aggregate.Type.COUNT, group,
CMObjects.all);

		ArrayList<Scalar> selectlist = new ArrayList<Scalar>();
		selectlist.add(count);
		Select select = new Select("", selectlist, group);

		EsperGenerator esper = new EsperGenerator();
		select.accept(esper);
		return esper;
	}
}
```

Output:

```
SELECT COUNT(*) FROM acimqp1.win:keepall()

	WHERE acimqp1.vInt2 < 1000

	AND acimqp1.vFloat1 < 50.0

	GROUP BY vInt2
```

Testing6:

```
public class Testing6 {

	@Test
	public void test() {
		String expected = "SELECT COUNT(*) FROM acimqp1.win:keepall() "
					+ "\n\tWHERE acimqp1.vInt1 < 1000"
					+ "\n\tORDER BY vInt1 ASC ";
		EsperGenerator esper = genEsper();
		esper.Print();
		assertEquals(expected, esper.getEsper());
	}

	public static EsperGenerator genEsper() {
		RawInteger md_tran_amtcomp = new RawInteger(1000);
		Comparison comp1 = new Comparison(Comparison.Type.LESS,
CMObjects.e1Int1, md_tran_amtcomp);
		Filter filter = new Filter(comp1, CMObjects.event1);
		EventSet order = new Order(filter,"vInt1","asc");

		Aggregate count = new Aggregate(Aggregate.Type.COUNT, order,
CMObjects.all);

		ArrayList<Scalar> selectlist = new ArrayList<Scalar>();
		selectlist.add(count);
		Select select = new Select("Shortwindow", selectlist, order);
		EsperGenerator esper = new EsperGenerator();
```

```
			select.accept(esper);
			return esper;
		}
}
```

Output:

```
SELECT COUNT(*) FROM acimqp1.win:keepall()

	WHERE acimqp1.vInt1 < 1000

	ORDER BY vInt1 ASC
```

Testing7:

```
public class Testing7 {

	@Test
	public void test() {
		String expected = "SELECT COUNT("
					+ "\n\tCASE 1"
					+ "\n\t\tWHEN 1 THEN 10"
					+ "\n\t\tWHEN 2 THEN 20"
					+ "\n\t\tELSE 999"
					+ "\n\tEND"
					+ "\n\t) FROM acimqp1.win:keepall().win:time(1 sec)"
					+ "\n\tWHERE acimqp1.vInt1 < 1000"
					+ "\n\tORDER BY vInt1 ASC ";
		EsperGenerator esper = genEsper();
		esper.Print();
		assertEquals(expected, esper.getEsper());
	}

	public static EsperGenerator genEsper() {
		RawInteger md_tran_amtcomp = new RawInteger(1000);
		Attribute atdd_date = new Attribute("variable", "DD_DATE");

		//DATEADD
		RawString SS = new RawString("SS");
		RawInteger neg_1 = new RawInteger(-1);
		RawString SHORTWINDOW_DD_DATE = new
RawString("SHORTWINDOW.DD_DATE");

		Scalar[] arguments = new Scalar[] {SS, neg_1, SHORTWINDOW_DD_DATE};
		List<Scalar> argslist = new ArrayList<Scalar>(Arrays.asList(arguments));
		DateOperation DATEADD = new
DateOperation(DateOperation.Name.DATEADD,argslist);

		//CASE
		RawInteger case_wh_1 = new RawInteger(1);
		RawString case_th_1 = new RawString("10");

		RawInteger case_wh_2 = new RawInteger(2);
		RawString case_th_2 = new RawString("20");

		RawString case_else = new RawString("999");

		CaseInstance CI_1 = new CaseInstance(case_wh_1,case_th_1);
		CaseInstance CI_2 = new CaseInstance(case_wh_2,case_th_2);
		CaseInstance CI_else = new CaseInstance(null, case_else);

		ArrayList<CaseInstance> CI_List = new ArrayList<CaseInstance>();
		CI_List.add(CI_1);
		CI_List.add(CI_2);
		CI_List.add(CI_else);

		RawInteger case_op = new RawInteger(1);
		Case case_1 = new Case(case_op,CI_List);

		Comparison comp1 = new Comparison(Comparison.Type.LESS,
CMObjects.e1Int1, md_tran_amtcomp);
		Comparison comp2 = new Comparison(Comparison.Type.GREATER, atdd_date,
DATEADD);

		LogicalOperation op1 = new LogicalOperation(LogicalOperation.Type.OR,
comp1, comp2);

		Filter filter = new Filter(op1, CMObjects.event1);
		EventSet order = new Order(filter,"vInt1","asc");
		Aggregate count = new Aggregate(Aggregate.Type.COUNT, order, case_1);

		ArrayList<Scalar> selectlist = new ArrayList<Scalar>();
		selectlist.add(count);
		Select select = new Select("", selectlist, order);
		EsperGenerator esper = new EsperGenerator();
		select.accept(esper);
		return esper;
	}
}
```

Output:

```
SELECT COUNT(
        CASE 1
                WHEN 1 THEN 10
                WHEN 2 THEN 20
                ELSE 999
        END
        ) FROM acimqp1.win:keepall().win:time(1 sec)
        WHERE acimqp1.vInt1 < 1000
        ORDER BY vInt1 ASC
```

Testing Nested Query:

```
public class TestNested1 {
        /*
         * SQL:
         * SELECT COUNT(*) FROM
         *       (SELECT vString1 FROM ACIMQP1 WHERE vString1 = 'US')
         *       WHERE vInt1 >= 300
         *       AND vInt1 <= 500
         */
        @Test
        public void test() {
                String expected = "SELECT COUNT(*) FROM acimqp1(vString1"
                                + "\n\tWHERE vString1 = 'US')"
                                + "\n\tWHERE vInt1 >= 300"
                                + "\n\tAND vInt1 <= 500";
                EsperGenerator esper = genEsper();
                esper.Print();
                assertEquals(expected, esper.getEsper());
        }

        public EsperGenerator genEsper() {
                //select count(*) from check.win:time(1 hours) where checkID > 450 and checkID
< 500
                RawInteger ri300 = new RawInteger(300);
                RawInteger ri500 = new RawInteger(500);

                Comparison comp1 = new Comparison(Comparison.Type.GREATEREQUALS,
CMObjects.vInt1, ri300);
```

```
		Comparison comp2 = new Comparison(Comparison.Type.LESSEQUALS,
CMObjects.vInt1, ri500);
		LogicalOperation op1 = new LogicalOperation(LogicalOperation.Type.AND,
comp1, comp2);

		Select nested1 = genSelect1();
		Filter filter = new Filter(op1, nested1);
		Aggregate count = new Aggregate(Aggregate.Type.COUNT, CMObjects.event1,
CMObjects.all);
		ArrayList<Scalar> selectlist = new ArrayList<Scalar>();
		selectlist.add(count);
		Select select = new Select("", selectlist, filter);

		EsperGenerator esper = new EsperGenerator();
		select.accept(esper);
		return esper;
	}

	public Select genSelect1() {
		Comparison comp1 = new Comparison(Comparison.Type.EQUALS,
CMObjects.vString1, CMObjects.rUS);

		Filter filter = new Filter(comp1, CMObjects.event1);
		ArrayList<Scalar> selectlist = new ArrayList<Scalar>();
		selectlist.add(CMObjects.vString1);
		Select select = new Select("", selectlist, filter);

		EsperGenerator esper = new EsperGenerator();
		select.accept(esper);
		return select;
	}
}
```

Output:

```
SELECT COUNT(*) FROM acimqp1(vString1

	WHERE vString1 = 'US')

	WHERE vInt1 >= 300

	AND vInt1 <= 500
```

9.3 APPENDIX II: VERIFYING ESPER SYNTAX

TestTesing2(with the output from Testing2):

```
public class TestTesting2 {
public static class Event {
        String SD_Term_Country;
        int SD_Term_ID;
        int MD_Tran_Amt1;

        public Event(int sdtc, int sdti, int md) {
                this.SD_Term_Country = "US";
                this.SD_Term_ID = sdti;
                this.MD_Tran_Amt1 = md;
        }
        public String getSD_Term_Country() {return this.SD_Term_Country;}
        public int getSD_Term_ID() {return this.SD_Term_ID;}
        public int getMD_Tran_Amt1() {return this.MD_Tran_Amt1;}
        @Override
        public String toString() {
                return "SD_Term_Country: " + this.SD_Term_Country + " SD_Term_ID:
" + this.SD_Term_ID + " MD_Tran_Amt1: " + this.MD_Tran_Amt1;
        }
}

private static Random generator = new Random();

public static void GenerateRandomTick(EPRuntime cepRT,int i) {

        int sdtc = generator.nextInt(2);
        int sdid = generator.nextInt(2);
        int md = generator.nextInt(2000);
        Event e = new Event(sdtc, sdid, md);
        System.out.println("Sending event: " + e);
        cepRT.sendEvent(e);
}
public static class CEPListener implements UpdateListener {
        public void update(EventBean[] newData, EventBean[] oldData) {
                System.out.println("Event received: " + newData[0].getUnderlying());
        }
}
public static void main(String[] args) throws InterruptedException {
        //The Configuration is meant only as an initialization-time object.
        Configuration cepConfig = new Configuration();
        cepConfig.addEventType("Shortwindow", Event.class.getName());
        EPServiceProvider cep =
EPServiceProviderManager.getProvider("myCEPEngine", cepConfig);
        EPRuntime cepRT = cep.getEPRuntime();
        EPAdministrator cepAdm = cep.getEPAdministrator();

        EPStatement cepStatement1 = cepAdm.createEPL(Testing2.genEsper().Print());
        cepStatement1.addListener(new CEPListener());

        // Generate a few event
        for (int i = 0; i < 10; i++) {
                GenerateRandomTick(cepRT,i);
                Thread.sleep(500);;
        }
}
}
```

Output :

```
SELECT COUNT(*) FROM Shortwindow.win:keepall().win:time(72 days)
        WHERE Shortwindow.MD_Tran_Amt1 < 1000
        AND Shortwindow.SD_Term_Country = 'US'
Sending event: SD_Term_Country: US SD_Term_ID: 1 MD_Tran_Amt1: 1429
Sending event: SD_Term_Country: US SD_Term_ID: 1 MD_Tran_Amt1: 1834
Sending event: SD_Term_Country: US SD_Term_ID: 1 MD_Tran_Amt1: 74
Event received: {count(*)=1}
Sending event: SD_Term_Country: US SD_Term_ID: 1 MD_Tran_Amt1: 1776
Sending event: SD_Term_Country: US SD_Term_ID: 1 MD_Tran_Amt1: 157
Event received: {count(*)=2}
Sending event: SD_Term_Country: US SD_Term_ID: 0 MD_Tran_Amt1: 55
Event received: {count(*)=3}
Sending event: SD_Term_Country: US SD_Term_ID: 0 MD_Tran_Amt1: 12
Event received: {count(*)=4}
Sending event: SD_Term_Country: US SD_Term_ID: 1 MD_Tran_Amt1: 1851
Sending event: SD_Term_Country: US SD_Term_ID: 0 MD_Tran_Amt1: 499
Event received: {count(*)=5}
Sending event: SD_Term_Country: US SD_Term_ID: 0 MD_Tran_Amt1: 1712
```

TestTesting3(with the output of Testing3):

```
        public class TestTesting3 {
        public static class Event1 {
                String SD_Term_Country;
                int SD_Term_ID;
                int MD_Tran_Amt1;
```

```
		public Event1(int sdtc, int sdti, int md) {
			this.SD_Term_Country = "US";
			this.SD_Term_ID = sdti;
			this.MD_Tran_Amt1 = md;
		}
		public String getSD_Term_Country() {return this.SD_Term_Country;}
		public int getSD_Term_ID() {return this.SD_Term_ID;}
		public int getMD_Tran_Amt1() {return this.MD_Tran_Amt1;}
		@Override
		public String toString() {
			return "SD_Term_Country: " + this.SD_Term_Country + " SD_Term_ID: 
" + this.SD_Term_ID + " MD_Tran_Amt1: " + this.MD_Tran_Amt1;
		}
	}
		public static class Event2 {
			String SD_Term_Country;
			int SD_Term_ID;
			int MD_Tran_Amt1;

			public Event2(int sdtc, int sdti, int md) {
				this.SD_Term_Country = "US";
				this.SD_Term_ID = sdti;
				this.MD_Tran_Amt1 = md;
			}
			public String getSD_Term_Country() {return this.SD_Term_Country;}
			public int getSD_Term_ID() {return this.SD_Term_ID;}
			public int getMD_Tran_Amt1() {return this.MD_Tran_Amt1;}
			@Override
			public String toString() {
				return "SD_Term_Country: " + this.SD_Term_Country + " 
SD_Term_ID: " + this.SD_Term_ID + " MD_Tran_Amt1: " + this.MD_Tran_Amt1;
			}
	}

	private static Random generator = new Random();
	public static void GenerateRandomTick(EPRuntime cepRT,int i) {

		int sdtc = generator.nextInt(2);
		int sdid = generator.nextInt(2);
		int md = generator.nextInt(700);
		Event1 e = new Event1(sdtc, sdid, md);
		System.out.println("Sending event: " + e);
		cepRT.sendEvent(e);
	}
	public static class CEPListener implements UpdateListener {
		public void update(EventBean[] newData, EventBean[] oldData) {
			System.out.println("Event received: " + newData[0].getUnderlying());
		}
	}
	public static void main(String[] args) throws InterruptedException {
		//The Configuration is meant only as an initialization-time object.
		Configuration cepConfig1 = new Configuration();
		cepConfig1.addEventType("Table1", Event1.class.getName());
		cepConfig1.addEventType("Table2", Event2.class.getName());
		EPServiceProvider cep1 =
EPServiceProviderManager.getProvider("myCEPEngine", cepConfig1);
		EPRuntime cepRT1 = cep1.getEPRuntime();
		EPAdministrator cepAdm1 = cep1.getEPAdministrator();
		EPStatement cepStatement1 = cepAdm1.createEPL(Testing3.genEsper().Print());
		cepStatement1.addListener(new CEPListener());

		// Generate a few event
		for (int i = 0; i < 10; i++) {
			GenerateRandomTick(cepRT1,i);
			//GenerateRandomTick(cepRT2,i);
			Thread.sleep(500);;
		}
	}
}
```
Output:

```
SELECT COUNT(*) FROM Table1.win:keepall() INNER JOIN
Table2.win:keepall().win:time(12 days) ON Table1.MD_Tran_Amt1 = Table2.MD_Tran_Amt1
	WHERE Table1.SD_Term_Country = 'US'
	OR Table1.SD_Term_Country = 'CA'
Sending event: SD_Term_Country: US SD_Term_ID: 1 MD_Tran_Amt1: 17
Sending event: SD_Term_Country: US SD_Term_ID: 1 MD_Tran_Amt1: 562
Sending event: SD_Term_Country: US SD_Term_ID: 0 MD_Tran_Amt1: 343
Sending event: SD_Term_Country: US SD_Term_ID: 0 MD_Tran_Amt1: 108
Sending event: SD_Term_Country: US SD_Term_ID: 0 MD_Tran_Amt1: 580
Sending event: SD_Term_Country: US SD_Term_ID: 0 MD_Tran_Amt1: 573
Sending event: SD_Term_Country: US SD_Term_ID: 1 MD_Tran_Amt1: 29
Sending event: SD_Term_Country: US SD_Term_ID: 1 MD_Tran_Amt1: 375
Sending event: SD_Term_Country: US SD_Term_ID: 1 MD_Tran_Amt1: 381
Sending event: SD_Term_Country: US SD_Term_ID: 1 MD_Tran_Amt1: 322
```

9.4 APPENDIX IV: TESTING THE TRANSFORMATION OF THE CANONICAL MODEL TO JAVA

TestRule3:

```
public class TestRule3 {

	/*
	 * Original:
	 *
	  IF ((SELECT COUNT(*) AS A FROM Shortwindow) > 1)
				AND (true = (@p1 = 1)) THEN
			PRINT 'Okay'
	  ELSE
			PRINT 'False'

	 */


	@Test
	public void test() {
		String expected =
					"EPServiceProvider epService = 
EPServiceProviderManager.getDefaultProvider();"
					+ "\nString EsperStatement = \"SELECT COUNT(*) FROM 
Shortwindow.win:keepall() \";"
					+ "\nEPStatement FEATURE = 
epService.getEPAdministrator().createEPL(EsperStatement);"
					+ "\npublic class MyListener implements UpdateListener {"
							+ "\npublic void update(EventBean[] newEvents, 
EventBean[] oldEvents) {"
							+ "\nif((newEvents[0].get(\"COUNT(*)\") > 1)){"
								+ "\n\tSystem.out.println(\"Okay\");"
								+"\n}"
								+ "\nelse{"
								+ "\n\tSystem.out.println(\"False\");"
								+ "\n}"
								+ "\n}"
								+ "\n}"
								+ "\nFEATURE.addListener(new 
MyListener());";

		assertEquals(genJava().Print(),expected);
	}

	public static javaGenerator genJava() {
```

```
                    CreateAlert iftrue = new
CreateAlert(CreateAlert.AlertType.PRINT,"Okay","Event1");
                    CreateAlert iffalse = new
CreateAlert(CreateAlert.AlertType.PRINT,"False","Event1");

                    RawInteger int_1 = new RawInteger(1);

                    EventSet events = new EventSet("Shortwindow");
                    Aggregate count = new Aggregate(Aggregate.Type.COUNT, events, new
RawString("*"));

                    Filter filter = new Filter(null, events);
                    ArrayList<Scalar> selectlist = new ArrayList<Scalar>();
                    selectlist.add(count);
                    Scalar select = new GetSingleCell(new Select("Shortwindow", selectlist, filter));

                    Comparison comp1 = new
Comparison(Comparison.Type.GREATER,select,int_1);

                    Rule rule = new Rule(iftrue,iffalse,comp1);

                    javaGenerator java = new javaGenerator();
                    rule.accept(java);
                    return java;
          }
}
```

Output:

```
EPServiceProvider epService = EPServiceProviderManager.getDefaultProvider();
String EsperStatement = "SELECT COUNT(*) FROM Shortwindow.win:keepall() ";
EPStatement FEATURE = epService.getEPAdministrator().createEPL(EsperStatement);
public class MyListener implements UpdateListener {
          public void update(EventBean[] newEvents, EventBean[] oldEvents) {
                    if((newEvents[0].get("COUNT(*)") > 1)){
                              System.out.println("Okay");
                    }
                    else{
                              System.out.println("False");
                    }
          }
}
FEATURE.addListener(new MyListener());
```

TestRule4:

```
public class TestRule4 {

	/*
	 * Original:
	 *
	  IF ((SELECT COUNT(*) AS A FROM Shortwindow) > 1)
			AND (true = (@p1 = 1)) THEN
		PRINT 'Okay'
	  ELSE
		PRINT 'False'

	 */

	@Test
	public void test() {
		String expected =
				"EPServiceProvider epService =
EPServiceProviderManager.getDefaultProvider();"
				+ "\nString EsperStatement = \"SELECT COUNT(*) AS A FROM
Shortwindow.win:keepall() \";"
				+ "\nEPStatement FEATURE =
epService.getEPAdministrator().createEPL(EsperStatement);"
				+ "\npublic class MyListener implements UpdateListener {"
						+ "\npublic void update(EventBean[] newEvents,
EventBean[] oldEvents) {"
						+ "\nif((newEvents[0].get(\"A\") > 1 + 0.5) && (true
== (@p1 == 1))){"
							+ "\n\tSystem.out.println(\"Okay\");"
							+"\n}"
							+ "\nelse{"
							+ "\n\tSystem.out.println(\"False\");"
							+ "\n}"
							+ "\n}"
							+ "\n}"
							+ "\nFEATURE.addListener(new
MyListener());";

		assertEquals(genJava().Print(),expected);
	}

	public static javaGenerator genJava() {

		CreateAlert iftrue = new
CreateAlert(CreateAlert.AlertType.PRINT,"Okay","Event1");
                    CreateAlert iffalse = new
CreateAlert(CreateAlert.AlertType.PRINT,"False","Event1");

                    RawInteger int_1 = new RawInteger(1);
                    RawFloat flt_1 = new RawFloat((float)0.5);
                    RawBoolean tr = new RawBoolean(true);

                    ArithmeticOperation AO1 = new
ArithmeticOperation(ArithmeticOperation.Operation.PLUS,int_1,flt_1);

                    EventSet events = new EventSet("Shortwindow");
                    Aggregate count = new Aggregate(Aggregate.Type.COUNT, events, new
RawString("*"));

                    Filter filter = new Filter(null, events);

                    Variable p1 = new Variable("@p1",Variable.Type.INT);

                    AliasColumn AC = new AliasColumn(count,"A");

                    ArrayList<Scalar> selectlist = new ArrayList<Scalar>();
                    selectlist.add(AC);
                    Scalar select = new GetSingleCell(new Select("Shortwindow", selectlist, filter));

                    Comparison comp1 = new
Comparison(Comparison.Type.GREATER,select,AO1);
                    Comparison comp2 = new Comparison(Comparison.Type.EQUALS,p1,int_1);
                    Comparison comp3 = new Comparison(Comparison.Type.EQUALS,tr,comp2);
                    LogicalOperation lop1 = new
LogicalOperation(LogicalOperation.Type.AND,comp1,comp3);

                    Rule rule = new Rule(iftrue,iffalse,lop1);

                    javaGenerator java = new javaGenerator();
                    rule.accept(java);
                    return java;
          }

}
```

Output:

```
EPServiceProvider epService = EPServiceProviderManager.getDefaultProvider();
String EsperStatement = "SELECT COUNT(*) AS A FROM Shortwindow.win:keepall() ";
EPStatement FEATURE = epService.getEPAdministrator().createEPL(EsperStatement);
public class MyListener implements UpdateListener {
        public void update(EventBean[] newEvents, EventBean[] oldEvents) {
                if((newEvents[0].get("A") > 1 + 0.5) && (true == (@p1 == 1))){
                        System.out.println("Okay");
                }
                else{
                        System.out.println("False");
                }
        }
}
        FEATURE.addListener(new MyListener());
```

TestRule5:

```
public class TestRule5 {

        /*
         * Original:
                IF  ( (SELECT COUNT(*) FROM Shortwindow) > 1)
                        AND ( (SELECT AVG(SD_KEY) FROM Shortwindow) <> 0)  THEN
                        PRINT N 'Okay';
                ELSE
                        PRINT N 'False';
                END

         */


        @Test
        public void test() {
                String expected =
                                "EPServiceProvider epService =
EPServiceProviderManager.getDefaultProvider();"
                                + "\nString EsperStatement = \"SELECT AVG(SD_KEY),
COUNT(*) FROM Shortwindow.win:keepall() \";"
                                + "\nEPStatement FEATURE =
epService.getEPAdministrator().createEPL(EsperStatement);"
                                + "\npublic class MyListener implements UpdateListener {"
                                        + "\npublic void update(EventBean[] newEvents,
EventBean[] oldEvents) {"
                                        + "\nif((newEvents[0].get(\"COUNT(*)\") > 1) &&
(newEvents[0].get(\"AVG(SD_KEY)\") != 0)){"
                                                + "\n\tSystem.out.println(\"Okay\");"
                                                +"\n}"
```

```
                                                        + "\nelse{"
                                                        + "\n\tSystem.out.println(\"False\");"
                                                        + "\n}"
                                                        + "\n}"
                                                        + "\n}"
                                                        + "\nFEATURE.addListener(new
MyListener());";

                assertEquals(genJava().Print(),expected);
        }

        public static javaGenerator genJava() {

                CreateAlert iftrue = new
CreateAlert(CreateAlert.AlertType.PRINT,"Okay","Event1");
                CreateAlert iffalse = new
CreateAlert(CreateAlert.AlertType.PRINT,"False","Event1");

                RawInteger int_1 = new RawInteger(1);
                RawInteger int_0 = new RawInteger(0);
                ProjectAll all = new ProjectAll(null);
                Attribute sdkey = new Attribute("SD_KEY");

                EventSet events = new EventSet("Shortwindow");
                Aggregate count = new Aggregate(Aggregate.Type.COUNT, events, all);
                Aggregate avg = new Aggregate(Aggregate.Type.AVG, events, sdkey);

                Filter filter = new Filter(null, events);

                ArrayList<Scalar> selectlist1 = new ArrayList<Scalar>();
                selectlist1.add(count);

                ArrayList<Scalar> selectlist2 = new ArrayList<Scalar>();
                selectlist2.add(avg);

                Scalar select1 = new GetSingleCell(new Select("Shortwindow", selectlist1,
filter));
                Scalar select2 = new GetSingleCell(new Select("Shortwindow", selectlist2,
filter));

                Comparison comp1 = new
Comparison(Comparison.Type.GREATER,select1,int_1);
                Comparison comp2 = new
Comparison(Comparison.Type.NOTEQUALS,select2,int_0);

                LogicalOperation lop1= new
```

```
LogicalOperation(LogicalOperation.Type.AND,comp1,comp2);

                Rule rule = new Rule(iftrue,iffalse,lop1);

                javaGenerator java = new javaGenerator();
                rule.accept(java);
                return java;
        }

}
```

Output:

```
EPServiceProvider epService = EPServiceProviderManager.getDefaultProvider();
String EsperStatement = "SELECT AVG(SD_KEY), COUNT(*) FROM
        Shortwindow.win:keepall() ";
EPStatement FEATURE = epService.getEPAdministrator().createEPL(EsperStatement);
public class MyListener implements UpdateListener {
        public void update(EventBean[] newEvents, EventBean[] oldEvents) {
                if((newEvents[0].get("COUNT(*)") > 1) &&
        (newEvents[0].get("AVG(SD_KEY)") != 0)){
                        System.out.println("Okay");
                }
                else{
                        System.out.println("False");
                }
        }
}
FEATURE.addListener(new MyListener());
```

TestRule6:

```
public class TestRule6 {

        /*
         * Original:
                IF  ( (SELECT COUNT(*) FROM Shortwindow)
                        > (SELECT AVG(SD_KEY) FROM Shortwindow)  THEN
                        PRINT 'Okay';
                ELSE
                        IF( ( (SELECT SUM(AMT) FROM Shortwindow) > 1000)
                        OR ( (SELECT COUNTRY FROM Shortwindow) = 'US') THEN
                                PRINT 'Okay';
```

```
				ELSE
					PRINT 'False';

			END

	 */



	@Test
	public void test() {
		String expected =
				"EPServiceProvider epService =
EPServiceProviderManager.getDefaultProvider();"
				+ "\nString EsperStatement = \"SELECT AVG(SD_KEY),
SUM(AMT), COUNTRY, COUNT(*) FROM Shortwindow.win:keepall() \";"
				+ "\nEPStatement FEATURE =
epService.getEPAdministrator().createEPL(EsperStatement);"
				+ "\npublic class MyListener implements UpdateListener {"
					+ "\npublic void update(EventBean[] newEvents,
EventBean[] oldEvents) {"
					+ "\nif((newEvents[0].get(\"COUNT(*)\") >
newEvents[0].get(\"AVG(SD_KEY)\"))){"
						+ "\n\tSystem.out.println(\"Okay\");"
						+"\n}"
						+ "\nelse{"
						+ "\n\tif((newEvents[0].get(\"SUM(AMT)\")
> 1000) || (newEvents[0].get(\"COUNTRY\") == \"US\")){"
						+ "\n\tSystem.out.println(\"Okay\");"
						+ "\n}"
						+ "\nelse{"
						+ "\n\tSystem.out.println(\"False\");"
						+ "\n}"
						+ "\n}"
						+ "\n}"
						+ "\n}"
						+ "\nFEATURE.addListener(new
MyListener());";

		assertEquals(genJava().Print(),expected);
	}
	/*
	public static void main(String[] args) {
		genJava().Print();
	}
	*/
	public static javaGenerator genJava() {

		CreateAlert iftrue = new
CreateAlert(CreateAlert.AlertType.PRINT,"Okay","Event1");
		CreateAlert iffalse = new
CreateAlert(CreateAlert.AlertType.PRINT,"False","Event1");

		RawInteger int_1000 = new RawInteger(1000);
		RawString us = new RawString("\"US\"");

		ProjectAll all = new ProjectAll(null);
		Attribute sdkey = new Attribute("SD_KEY");
		Attribute amt = new Attribute("AMT");
		Attribute country = new Attribute("COUNTRY");

		EventSet events = new EventSet("Shortwindow");
		Aggregate count = new Aggregate(Aggregate.Type.COUNT, events, all);
		Aggregate avg = new Aggregate(Aggregate.Type.AVG, events, sdkey);
		Aggregate sum = new Aggregate(Aggregate.Type.SUM,events,amt);
		//Aggregate max = new Aggregate(Aggregate.Type.FIRST,events,country);

		Filter filter = new Filter(null, events);

		ArrayList<Scalar> selectlist1 = new ArrayList<Scalar>();
		selectlist1.add(count);

		ArrayList<Scalar> selectlist2 = new ArrayList<Scalar>();
		selectlist2.add(avg);

		ArrayList<Scalar> selectlist3 = new ArrayList<Scalar>();
		selectlist3.add(sum);

		ArrayList<Scalar> selectlist4 = new ArrayList<Scalar>();
		selectlist4.add(country);


		Scalar select1 = new GetSingleCell(new Select("Shortwindow", selectlist1,
filter));
		Scalar select2 = new GetSingleCell(new Select("Shortwindow", selectlist2,
filter));
		Scalar select3 = new GetSingleCell(new Select("Shortwindow", selectlist3,
filter));
		Scalar select4 = new GetSingleCell(new Select("Shortwindow", selectlist4,
filter));

		Comparison comp1 = new
Comparison(Comparison.Type.GREATER,select1,select2);
                    Comparison comp2 = new
Comparison(Comparison.Type.GREATER,select3,int_1000);
                    Comparison comp3 = new Comparison(Comparison.Type.EQUALS,select4,us);

                    LogicalOperation lop1 = new
LogicalOperation(LogicalOperation.Type.OR,comp2,comp3);

                    Rule rule2 = new Rule(iftrue,iffalse,lop1);
                    Rule rule1 = new Rule(iftrue,rule2,comp1);


                    javaGenerator java = new javaGenerator();
                    rule1.accept(java);
                    return java;
          }

}
```

Output:

```
EPServiceProvider epService = EPServiceProviderManager.getDefaultProvider();
String EsperStatement = "SELECT AVG(SD_KEY), SUM(AMT), COUNTRY, COUNT(*)
          FROM Shortwindow.win:keepall() ";
EPStatement FEATURE = epService.getEPAdministrator().createEPL(EsperStatement);
public class MyListener implements UpdateListener {
          public void update(EventBean[] newEvents, EventBean[] oldEvents) {
                    if((newEvents[0].get("COUNT(*)") > newEvents[0].get("AVG(SD_KEY)"))){
                              System.out.println("Okay");
                    }
                    else{
                              if((newEvents[0].get("SUM(AMT)") > 1000) ||
                                                  (newEvents[0].get("COUNTRY") == "US")){
                                        System.out.println("Okay");
                              }
                              else{
                                        System.out.println("False");
                              }
                    }
          }
}
FEATURE.addListener(new MyListener());
```

9.5 Appendix V: Testing The Transformation of SQL Feature to Esper Statements

```
package test.visitors;
import static org.junit.Assert.*;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import org.junit.Test;
import visitors.ParseParser;
public class ParseParserTest {
	static int i = 1;
	public void writeToFile(String content){
		try {
			File file = new File(System.getProperty("user.home") + "/workspace/ACI
MQP/test.sql");
			// if file doesnt exists, then create it
			if (!file.exists()) {
				file.createNewFile();
			}
			FileWriter fw = new FileWriter(file.getAbsoluteFile());
			BufferedWriter bw = new BufferedWriter(fw);
			bw.write(content);
			bw.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
	public void startTest(){
		System.out.println("---- STARTING TEST " + i + " ----");
		System.out.println();
	}
	public void endTest(){
		System.out.println();
		System.out.println("------ DONE TEST " + i + " ------");
		System.out.println();
		i++;
	}
	@Test
	public void testVeryComplex1() {
		startTest();
		writeToFile("SELECT c = CASE\n WHEN count(transactions) > 1 THEN 1\n
WHEN count(transactions) < 1 THEN 0\n END\n FROM table\n" +
				"WHERE (SELECT max(amt) FROM stream) > 100");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT \n" + "\tCASE \n" + "\t\tWHEN
COUNT(transactions) > 1 THEN 1\n" +
				"\t\tWHEN COUNT(transactions) < 1 THEN 0\n" + "\tEND\n" +
"\t AS c FROM table.win:keepall() \n" +
				"\tWHERE  ( SELECT MAX(amt) FROM stream.win:keepall() )
> 100");
		endTest();
	}


	@Test
	public void testVeryComplex2() {
		startTest();
		writeToFile("SELECT \n" +
				"CASE RSLT\n" +
				"WHEN \"1\" THEN 0.163318\n" +
				"WHEN \"2\" THEN 0.190563\n" +
				"WHEN \"3\" THEN 0.0764718\n" +
				"WHEN \"4\" THEN 0.230415\n" +
				"WHEN \"5\" THEN 0.463535\n" +
				"WHEN \"6+\" THEN 1.4217\n" +
				"ELSE 0\n" +
				"END\n" +
				"FROM\n" +
				"(SELECT\n" +
				"RSLT = CASE (SELECT COUNT(DISTINCT(SD_KEY))+1
FROM SHORTWINDOW\n" +
				"WHERE SD_TERM_ID = SHORTWINDOW.SD_TERM_ID\n"
+
				"AND DD_DATE < SHORTWINDOW.DD_DATE\n" +
				"AND DD_DATE > DATEADD(MI, -10,
SHORTWINDOW.DD_DATE)\n" +
				"AND MD_TRAN_AMT1 >= 100\n" +
				"AND SD_KEY <> SHORTWINDOW.SD_KEY)\n" +
				"WHEN 1 THEN \"1\"\n" +
				"WHEN 2 THEN \"2\"\n" +
                    "WHEN 3 THEN \"3\"\n" +
                    "WHEN 4 THEN \"4\"\n" +
                    "WHEN 5 THEN \"5\"\n" +
                    "ELSE \"6+\"\n" +
                    "END)");
        ParseParser pp = new ParseParser();
        String[] args = {"test", "1"};
        ParseParser.main(args);
        String output = pp.getQuery();
        assertEquals(output, "SELECT \n" +
                    "\tCASE RSLT\n" +
                    "\t\tWHEN \"1\" THEN 0.163318\n" +
                    "\t\tWHEN \"2\" THEN 0.190563\n" +
                    "\t\tWHEN \"3\" THEN 0.0764718\n" +
                    "\t\tWHEN \"4\" THEN 0.230415\n" +
                    "\t\tWHEN \"5\" THEN 0.463535\n" +
                    "\t\tWHEN \"6+\" THEN 1.4217\n" +
                    "\t\tELSE 0\n" +
              "\tEND\n" +
              "\t FROM \n" +
              "\tCASE  ( shortwindow(COUNT() + 1\n" +
              "\tWHERE SD_TERM_ID = SHORTWINDOW.SD_TERM_ID\n" +
              "\tAND DD_DATE < SHORTWINDOW.DD_DATE\n" +
              "\tAND MD_TRAN_AMT1 >= 100\n" +
              "\tAND SD_KEY <> SHORTWINDOW.SD_KEY).win:time(10 min)\n"
+
                    "\t\tWHEN 1 THEN \"1\"\n" +
                    "\t\tWHEN 2 THEN \"2\"\n" +
                    "\t\tWHEN 3 THEN \"3\"\n" +
                    "\t\tWHEN 4 THEN \"4\"\n" +
                    "\t\tWHEN 5 THEN \"5\"\n" +
                    "\t\tELSE \"6+\"\n" +
                "\tEND\n" +
                "\t AS RSLT)");
        endTest();
    }

    @Test
    public void testSimpleQuery() {
        startTest();
        writeToFile("SELECT a FROM b");
        ParseParser pp = new ParseParser();
        String[] args = {"test", "1"};
        ParseParser.main(args);
        String output = pp.getQuery();
        assertEquals(output, "SELECT a FROM b.win:keepall() ");
        endTest();
    }

    @Test
    public void testSimpleQueryWithBool() {
        startTest();
        writeToFile("SELECT true FROM b");
        ParseParser pp = new ParseParser();
        String[] args = {"test", "1"};
```

```
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT true FROM b.win:keepall() ");
		endTest();
	}

	@Test
	public void testNOT() {
		startTest();
		writeToFile("SELECT a FROM b WHERE NOT a > b");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE 
NOT a > b");
		endTest();
	}

	@Test
	public void testArithmeticOpQuery() {
		startTest();
		writeToFile("SELECT 10+count(b),count(b)/2,c%2,d&1,e|1,c^1,POWER(f,2) 
FROM b WHERE a+2>b-2 HAVING count(a*b) > 10 GROUP BY c");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT 10 + COUNT(b), COUNT(b) / 2, c % 2, d & 1, e | 1, c ^ 1 FROM b.win:keepall() " + "\n\t" + "WHERE a + 2 > b - 2" + "\n\t" + "GROUP BY c" + "\n\t" + "HAVING COUNT(a * b) > 10");

		endTest();

	}


	@Test

	public void testSimpleQueryWithMultipleSelections() {

		startTest();

		writeToFile("SELECT a, b FROM c");

		ParseParser pp = new ParseParser();

		String[] args = {"test", "1"};

		ParseParser.main(args);

		String output = pp.getQuery();

		assertEquals(output, "SELECT a, b FROM c.win:keepall() ");

		endTest();

	}


	@Test

	public void testGroupBy1() {

		startTest();

		writeToFile("SELECT count(a) FROM c GROUP BY a, c");

		ParseParser pp = new ParseParser();

		String[] args = {"test", "1"};

		ParseParser.main(args);

		String output = pp.getQuery();

		assertEquals(output, "SELECT COUNT(a) FROM c.win:keepall() " + "\n\t" + "GROUP BY a, c");
		endTest();
	}


	@Test
	public void testAggregateQuery() {
		startTest();
		writeToFile("SELECT COUNT(a), AVG(a), STDDEV(a), MIN(a), MAX(a), 
FIRST(a), LAST(a) FROM b");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT COUNT(a), AVG(a), STDDEV(a), MIN(a), 
MAX(a), FIRST(a), LAST(a) FROM b.win:keepall() ");
		endTest();
	}


	@Test
	public void testComplexQuery(){
		startTest();
		writeToFile("SELECT a FROM b WHERE a > b GROUP BY c ORDER By d 
DESC");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE a 
> b" + "\n\t" + "GROUP BY c" + "\n\t" + "ORDER BY d DESC ");
		endTest();
	}

	@Test
	public void testSimpleQueryWithWhere(){
		startTest();
		writeToFile("SELECT a FROM b WHERE c > 10");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE c
> 10");
		endTest();
	}

	@Test
	public void testSimpleQueryWithComplexWhereAND(){
		startTest();
		writeToFile("SELECT a FROM b WHERE c > 10 AND d < 100");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE c
> 10" + "\n\t" + "AND d < 100");
		endTest();
	}
	@Test
	public void testSimpleQueryWithComplexWhereOR(){
		startTest();
		writeToFile("SELECT a FROM b WHERE c > 10 OR d < 100");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE c > 10" + "\n\t" + "OR d < 100");
		endTest();
	}

	@Test
	public void testSimpleQueryWithHaving1(){
		startTest();
		writeToFile("SELECT a FROM b HAVING count(a) > 100");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "HAVING COUNT(a) > 100");
		endTest();
	}

	@Test
	public void testSimpleQueryWithHaving2(){
		startTest();
		writeToFile("SELECT a FROM b HAVING count(a) = 'bbb'");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "HAVING 
COUNT(a) = 'bbb'");
		endTest();
	}

	@Test
	public void testSimpleQueryWithHaving3(){
		startTest();
		writeToFile("SELECT a FROM b HAVING count(a) != bbb");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "HAVING 
COUNT(a) <> bbb");
		endTest();
	}

	@Test
	public void testSimpleQueryWithHaving4(){
		startTest();
		writeToFile("SELECT a FROM b HAVING count(a) <> a+1");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "HAVING
COUNT(a) <> a + 1");
		endTest();
	}

	@Test
	public void testSimpleQueryWithHaving5(){
		startTest();
		writeToFile("SELECT a FROM b HAVING count(a) >= sum(b)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "HAVING
COUNT(a) >= SUM(b)");
		endTest();
	}

	@Test
	public void testSimpleQueryWithHaving6(){
		startTest();
		writeToFile("SELECT a FROM b HAVING count(a) <= (select b from c)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "HAVING
COUNT(a) <=  ( SELECT b FROM c.win:keepall() ) ");
		endTest();
	}

	@Test
	public void testSimpleQueryWithHaving7(){
		startTest();
		writeToFile("SELECT a FROM b HAVING count(a) < (select b from c)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "HAVING
COUNT(a) <  ( SELECT b FROM c.win:keepall() ) ");
		endTest();
	}

	@Test
	public void testSimpleQueryWithGroupBy(){
		startTest();
		writeToFile("SELECT a FROM b GROUP BY c");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "GROUP
BY c");
		endTest();
	}

	@Test
	public void testSimpleQueryWithOrderByDESC(){
		startTest();
		writeToFile("SELECT a FROM b ORDER BY c DESC");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "ORDER
BY c DESC "); // TODO FIX
		endTest();
	}

	@Test
	public void testSimpleQueryWithOrderByASC(){
		startTest();
		writeToFile("SELECT b.a FROM b ORDER BY c ASC");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT b.a FROM b.win:keepall() " + "\n\t" + "ORDER
BY c ASC "); // TODO FIX
```

```
		endTest();
	}

	@Test
	public void testSimpleQueryWithSelectSubquery(){
		startTest();
		writeToFile("SELECT (SELECT aaa FROM stream) FROM b");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT  ( SELECT aaa FROM stream.win:keepall() ) 
FROM b.win:keepall() ");
		endTest();
	}

	@Test
	public void testSimpleQueryWithWhereSubquery(){
		startTest();
		writeToFile("SELECT a FROM b WHERE (SELECT COUNT(*) FROM c) > 
d");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE 
( SELECT COUNT(*) FROM c.win:keepall() )  > d");
		endTest();
	}


	@Test
	public void testCompleteSimpleQuery() {
		startTest();
		writeToFile("SELECT TOP 1 a FROM b WHERE c > 1 GROUP BY d HAVING
SUM(e) > 2 ORDER BY f DESC");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT TOP 1 a FROM b.win:keepall() " + "\n\t" +
"WHERE c > 1" + "\n\t" +
				"GROUP BY d" + "\n\t" + "HAVING SUM(e) > 2" + "\n\t" +
"ORDER BY f DESC ");
		endTest();
	}


	@Test
	public void testLimit(){
		startTest();
		writeToFile("SELECT TOP 1 a FROM b ");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT TOP 1 a FROM b.win:keepall() ");
		endTest();
	}


	@Test
	public void testSubqueryInFROM(){
		startTest();
		writeToFile("SELECT a FROM (SELECT c FROM b)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b(c)");
		endTest();
	}


	@Test
	public void testKeywordIN1(){
		startTest();
		writeToFile("SELECT a FROM b where c IN ('xx', a)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE c
IN ( 'xx', a) ");
		endTest();
	}


	@Test
	public void testKeywordIN2(){
		startTest();
		writeToFile("SELECT a FROM b where 2 IN ('xx', a, a+1)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE 2 
IN ( 'xx', a, a + 1) ");
		endTest();
	}

	@Test
	public void testKeywordIN3(){
		startTest();
		writeToFile("SELECT a FROM b where (select a from b) IN (select b from c)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE 
( SELECT a FROM b.win:keepall() )  IN (  ( SELECT b FROM c.win:keepall() ) ) ");
		endTest();
	}

	@Test
	public void testKeywordIN4(){
		startTest();
		writeToFile("SELECT a FROM b where c+1 IN (select b from c)");

		ParseParser pp = new ParseParser();

		String[] args = {"test", "1"};

		ParseParser.main(args);

		String output = pp.getQuery();

		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE c 
+ 1 IN (  ( SELECT b FROM c.win:keepall() ) ) ");

		endTest();

	}


	@Test

	public void testLike1(){

		startTest();

		writeToFile("SELECT a FROM b where c LIKE 's%'");

		ParseParser pp = new ParseParser();

		String[] args = {"test", "1"};

		ParseParser.main(args);

		String output = pp.getQuery();

		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE c 
LIKE 's%'");

		endTest();

	}


	@Test

	public void testLike2(){

		startTest();

		writeToFile("SELECT a FROM b where 10 LIKE a");

		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE 10 
LIKE a");
		endTest();
	}

	@Test
	public void testBetween1(){
		startTest();
		writeToFile("SELECT a FROM b where c BETWEEN 10 and 11");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE c 
>= 10" + "\n\t" + "AND c <= 11");
		endTest();
	}

	@Test
	public void testBetween2(){
		startTest();
		writeToFile("SELECT a FROM b where 10 BETWEEN c and b");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE 10 
>= c" + "\n\t" + "AND 10 <= b");
		endTest();
	}


	@Test
	public void testBetween3(){
		startTest();
		writeToFile("SELECT a FROM b where (select a from b) BETWEEN (select a 
from b) and (select a from b)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE 
( SELECT a FROM b.win:keepall() )  >=  ( SELECT a FROM b.win:keepall() ) "
					+ "\n\t" + "AND  ( SELECT a FROM b.win:keepall() )  <= 
( SELECT a FROM b.win:keepall() ) ");
		endTest();
	}


	@Test
	public void testBetween4(){
		startTest();
		writeToFile("SELECT a FROM b where a+1 BETWEEN b-1 and b+10");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE a
+ 1 >= b - 1" + "\n\t" + "AND a + 1 <= b + 10");
		endTest();
	}

	@Test
	public void testKeywordExist(){
		startTest();
		writeToFile("SELECT a FROM b where EXISTS (select * from b)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE
EXISTS ( SELECT * FROM b.win:keepall()  ) ");
		endTest();
	}

	@Test
	public void testProjectAll(){
		startTest();
		writeToFile("SELECT * FROM b");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT * FROM b.win:keepall() ");
		endTest();
	}

	@Test
	public void testAlias(){
		startTest();
		writeToFile("SELECT a AS b FROM c AS d");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a AS b FROM c.win:keepall() AS d ");
		endTest();
	}

	@Test
	public void testCase1(){
		startTest();
		writeToFile("SELECT CASE ccc WHEN a THEN c WHEN 10 THEN 1 END
FROM b");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT " + "\n\t" + "CASE ccc" + "\n\t\t" + "WHEN a
THEN c" + "\n\t\t" + "WHEN 10 THEN 1" + "\n\t" + "END" + "\n\t " + "FROM b.win:keepall()
");
		endTest();
	}


	@Test

	public void testCase2(){

		startTest();

		writeToFile("SELECT CASE WHEN a >= 0 THEN 0 WHEN b < 0 THEN 1 
END FROM b");

		ParseParser pp = new ParseParser();

		String[] args = {"test", "1"};

		ParseParser.main(args);

		String output = pp.getQuery();

		assertEquals(output, "SELECT " + "\n\t" + "CASE " + "\n\t\t" + "WHEN a >= 0 
THEN 0" + "\n\t\t" + "WHEN b < 0 THEN 1" + "\n\t" + "END" + "\n\t " + "FROM 
b.win:keepall() ");

		endTest();

	}


	@Test

	public void testCase3(){

		startTest();

		writeToFile("SELECT CASE WHEN a >= 'str' THEN 0 WHEN a < 0 THEN 'bbb' 
END FROM b");

		ParseParser pp = new ParseParser();

		String[] args = {"test", "1"};

		ParseParser.main(args);

		String output = pp.getQuery();

		assertEquals(output, "SELECT " + "\n\t" + "CASE " + "\n\t\t" + "WHEN a >= 
'str' THEN 0" + "\n\t\t" + "WHEN a < 0 THEN 'bbb'" + "\n\t" + "END" + "\n\t " + "FROM 
b.win:keepall() ");

		endTest();
	}


	@Test
	public void testCase4(){
		startTest();
		writeToFile("SELECT CASE 'ddd' WHEN 'ccc' THEN 'sss' WHEN 10 THEN 1
END FROM b");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT " + "\n\t" + "CASE 'ddd'" + "\n\t\t" + "WHEN
'ccc' THEN 'sss'" + "\n\t\t" + "WHEN 10 THEN 1" + "\n\t" + "END" + "\n\t " + "FROM
b.win:keepall() ");
		endTest();
	}


	@Test
	public void testAll1(){
		startTest();
		writeToFile("SELECT a FROM b WHERE a = ALL (select a from c)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE a
= ALL ( SELECT a FROM c.win:keepall() ) ");
		endTest();
	}
	@Test
	public void testAll2(){
		startTest();
		writeToFile("SELECT a FROM b WHERE 10 <> ALL (select a from c)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE 10
<> ALL ( SELECT a FROM c.win:keepall() ) ");
		endTest();
	}

	@Test
	public void testAll3(){
		startTest();
		writeToFile("SELECT a FROM b WHERE a+1 != ALL (select a from c)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE a
+ 1 <> ALL ( SELECT a FROM c.win:keepall() ) ");
		endTest();
	}

	@Test
	public void testAll4(){
		startTest();
		writeToFile("SELECT a FROM b WHERE (select a from b) > ALL (select a from
c)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE
( SELECT a FROM b.win:keepall() )  > ALL ( SELECT a FROM c.win:keepall() ) ");
		endTest();
	}

	@Test
	public void testAll5(){
		startTest();
		writeToFile("SELECT a FROM b WHERE a >= ALL (select a from c)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE a
>= ALL ( SELECT a FROM c.win:keepall() ) ");
		endTest();
	}

	@Test
	public void testAny6(){
		startTest();
		writeToFile("SELECT a FROM b WHERE a < ANY (select a from c)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE a
< ANY ( SELECT a FROM c.win:keepall() ) ");
		endTest();
	}


	@Test
	public void testSome1(){
		startTest();
		writeToFile("SELECT a FROM b WHERE a <= SOME (select a from c)");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() " + "\n\t" + "WHERE a
<= SOME ( SELECT a FROM c.win:keepall() ) ");
		endTest();
	}


	@Test
	public void testCrossJoin1(){
		startTest();
		writeToFile("SELECT a FROM b, c");
```

```
                ParseParser pp = new ParseParser();
                String[] args = {"test", "1"};
                ParseParser.main(args);
                String output = pp.getQuery();
                assertEquals(output, "SELECT a FROM b.win:keepall() CROSS JOIN
c.win:keepall() ");
                endTest();
        }


        @Test
        public void testInnerJoin1(){
                startTest();
                writeToFile("SELECT a FROM b INNER JOIN c ON b.d=c.d");
                ParseParser pp = new ParseParser();
                String[] args = {"test", "1"};
                ParseParser.main(args);
                String output = pp.getQuery();
                assertEquals(output, "SELECT a FROM b.win:keepall() INNER JOIN
c.win:keepall()  ON b.d = c.d");
                endTest();
        }


        @Test
        public void testNaturalJoin1(){
                startTest();
                writeToFile("SELECT a FROM b NATURAL JOIN c");
                ParseParser pp = new ParseParser();
                String[] args = {"test", "1"};
```

```
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() NATURAL JOIN
c.win:keepall() ");
		endTest();
	}

	@Test
	public void testLeftOuterJoin1(){
		startTest();
		writeToFile("SELECT a FROM b LEFT OUTER JOIN c ON b.d=c.d");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "SELECT a FROM b.win:keepall() LEFT OUTER JOIN
c.win:keepall()  ON b.d = c.d");
		endTest();
	}

	@Test
	public void testRightOuterJoin1(){
		startTest();
		writeToFile("SELECT a FROM b RIGHT OUTER JOIN c ON b.d=c.d");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
```

```
                assertEquals(output, "SELECT a FROM b.win:keepall() RIGHT OUTER JOIN 
c.win:keepall()  ON b.d = c.d");
                endTest();
        }


        @Test
        public void testFullOuterJoin1(){
                startTest();
                writeToFile("SELECT a FROM b FULL OUTER JOIN c ON b.d=c.d");
                ParseParser pp = new ParseParser();
                String[] args = {"test", "1"};
                ParseParser.main(args);
                String output = pp.getQuery();
                assertEquals(output, "SELECT a FROM b.win:keepall() FULL OUTER JOIN 
c.win:keepall()  ON b.d = c.d");
                endTest();
        }


        @Test
        public void testCrossJoin2(){
                startTest();
                writeToFile("SELECT a FROM b CROSS JOIN c");
                ParseParser pp = new ParseParser();
                String[] args = {"test", "1"};
                ParseParser.main(args);
                String output = pp.getQuery();
                assertEquals(output, "SELECT a FROM b.win:keepall() CROSS JOIN 
c.win:keepall() ");
```

```
		endTest();
	}

	@Test
	public void testSimpleSet() {
		startTest();
		writeToFile("DECLARE @myvar int;" + "\n" + "SET @myvar = 10;");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		endTest();
	}

}
```

9.6 APPENDIX VI: TESTING THE TRANSFORMATION OF SQL RULES TO JAVA CODE

```
package test.Rules;
import static org.junit.Assert.*;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import org.junit.Test;
import visitors.ParseParser;
public class TestParseParserRules {
	static int i = 1;
	public void writeToFile(String content){
		try {
			File file = new File(System.getProperty("user.home") + "/workspace/ACI
MQP/test.sql");
			// if file doesnt exists, then create it
			if (!file.exists()) {
				file.createNewFile();
			}
			FileWriter fw = new FileWriter(file.getAbsoluteFile());
			BufferedWriter bw = new BufferedWriter(fw);
			bw.write(content);
			bw.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public void startTest(){
		System.out.println("---- STARTING TEST " + i + " ----");
		System.out.println();
	}

	public void endTest(){
		System.out.println();
		System.out.println("------ DONE TEST " + i + " ------");
		System.out.println();
		i++;
	}
	@Test
	public void testSimpleRule() {
		startTest();
		writeToFile("IF (select COUNT(*) FROM table) > 100" + "\n" + "PRINT
'approve'" + "\n" + "ELSE" + "\n" + "PRINT 'decline'");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "EPServiceProvider epService =
EPServiceProviderManager.getDefaultProvider();" + "\n" +
				"String EsperStatement = \"SELECT COUNT(*) FROM
table.win:keepall() \";" + "\n" +
				"EPStatement FEATURE =
epService.getEPAdministrator().createEPL(EsperStatement);" + "\n" +
				"public class MyListener implements UpdateListener {" + "\n" +
				"public void update(EventBean[] newEvents, EventBean[]
oldEvents) {" + "\n" +
				"if((newEvents[0].get(\"COUNT(*)\") > 100)){" + "\n" +
				"\t" + "System.out.println(\"approve\");" + "\n" +
				"}\n" + "else{" + "\n" + "\tSystem.out.println(\"decline\");" +
"\n}\n" + "}\n" + "}\n" +
				"FEATURE.addListener(new MyListener());");
		endTest();
	}

	@SuppressWarnings("unused")
	@Test
	public void testInRule() {
		startTest();
		writeToFile("IF (select COUNT(*) FROM table) IN (100,200,300)" + "\n" + 
"PRINT 'approve'" + "\n" + "ELSE" + "\n" + "PRINT 'decline'");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		endTest();
	}


	@Test
	public void testSetRule() {
		startTest();
		writeToFile("IF @P1 = 0" + "\n" + "SET @x1 = 0" + "\n" + "ELSE" + "\n" + 
"SET @x1 = 1");
		ParseParser pp = new ParseParser();
		String[] args = {"test", "1"};
		ParseParser.main(args);
		String output = pp.getQuery();
		assertEquals(output, "EPServiceProvider epService = 
EPServiceProviderManager.getDefaultProvider();\n" +
				"String EsperStatement = \"SELECT\";\n" +
				"EPStatement FEATURE = 
epService.getEPAdministrator().createEPL(EsperStatement);\n" +
				"public class MyListener implements UpdateListener {\n" +
				"public void update(EventBean[] newEvents, EventBean[] 
oldEvents) {\n" +
				"if((@P1 == 0)){\n" +
				"\tx1 = 0;\n" +
                        "}\n" +
                        "else{\n" +
                        "\tx1 = 1;\n" +
                        "}\n" +
                        "}\n" +
                        "}\n" +
                        "FEATURE.addListener(new MyListener());");
            endTest();
    }


    @Test
    public void testLikeRule() {
            startTest();
            writeToFile("IF (select a FROM table) LIKE '%s'" + "\n" + "PRINT 'approve'");
            ParseParser pp = new ParseParser();
            String[] args = {"test", "1"};
            ParseParser.main(args);
            String output = pp.getQuery();
            assertEquals(output, "EPServiceProvider epService =
EPServiceProviderManager.getDefaultProvider();" + "\n" +
                        "String EsperStatement = \"SELECT a FROM table.win:keepall()
\";" + "\n" +
                        "EPStatement FEATURE =
epService.getEPAdministrator().createEPL(EsperStatement);" + "\n" +
                        "public class MyListener implements UpdateListener {" + "\n" +
                        "public void update(EventBean[] newEvents, EventBean[]
oldEvents) {" + "\n" +
                        "if(newEvents[0].get(\"a\").matches(\".*s\")){" + "\n" +
                        "\t" + "System.out.println(\"approve\");" + "\n" +
                                        "}\n" + "}\n" + "}\n" +

                                        "FEATURE.addListener(new MyListener());");

                endTest();

        }

}
```

9.7 APPENDIX VII: VERIFYING CORRECTNESS OF THE TRANSLATION

EventSetup: Set up the events ready for comparing the results of running SQL vs Esper against the database. This class will be used in TestTesting5, TestTesting6, TestTesting7

```
public class EventSetup {
        private Connection connection;
        private String sqlQuery;
        private String[] attributeList;
        private static List<EventBean> esperResult;

        /**
         * If an event pass the Esper query, process it here
         */
        public static class CEPListener implements UpdateListener {
                public void update(EventBean[] newData, EventBean[] oldData) {
                        System.out.println("Event received: " + newData[0].getUnderlying());
                        updateEsperResult(newData[0]);
                }
        }

        /**
         * @param event
         */
        public static void updateEsperResult(EventBean event) {
                esperResult.add(event);
        }

        /**
         * An EventSetup consists of a SQL connection, SQL query, and a list of "SELECT" 
attributes
         * @param conn
         * @param sql
         * @param attrList
         */
        public EventSetup(Connection conn, String sql, String[] attrList) {
                esperResult = new ArrayList<EventBean>();
```

```
		this.connection = conn;
		this.sqlQuery = sql;
		this.attributeList = attrList;
	}

	/**
	 * Compare the results of running SQL vs Esper against the database
	 * @return
	 * @throws SQLException
	 */
	public boolean compareResults() throws SQLException {
		boolean resultMatch = true;
		int i = 0;
		ResultSet sqlResult = DatabaseGenerator.executeStatement(this.connection,
sqlQuery);
		System.out.println("COMPARE RESULTS:");
		while (sqlResult.next()) {
			boolean rowMatch = true;
			for (int j = 0; j < attributeList.length; j++) {
				if (attributeList[j].contains("count(")) {
					long esperColumn = (long)
esperResult.get(esperResult.size()-1).get(attributeList[0]);
					resultMatch &= (esperColumn ==
sqlResult.getLong(attributeList[j].replaceAll(" ", "")));
					System.out.println(attributeList[j] + "=" + esperColumn +
"," + sqlResult.getLong(attributeList[j].replaceAll(" ", "")) + " | Matching=" + resultMatch);
				} else {
					Object esperColumn =
esperResult.get(i).get(attributeList[j]);
					if
(Integer.class.isAssignableFrom(esperColumn.getClass())) {
						rowMatch &= ((Integer) esperColumn ==
sqlResult.getInt(attributeList[j].replaceAll(" ", "")));
						System.out.print(attributeList[j] + "=" +
esperColumn + "," + sqlResult.getInt(attributeList[j]) + " ");
					} else if
(String.class.isAssignableFrom(esperColumn.getClass())) {
						rowMatch &=
(esperColumn.equals(sqlResult.getString(attributeList[j].replaceAll(" ", ""))));
						System.out.print(attributeList[j] + "=" +
esperColumn + "," + sqlResult.getString(attributeList[j]) + " ");
					} else if
(Float.class.isAssignableFrom(esperColumn.getClass())) {
						rowMatch &= ((Float) esperColumn ==
sqlResult.getFloat(attributeList[j].replaceAll(" ", "")));
```

```
                                                System.out.print(attributeList[j] + "=" +
esperColumn + "," + sqlResult.getFloat(attributeList[j]) + " ");
                                        } else if
(Date.class.isAssignableFrom(esperColumn.getClass())){
                                                rowMatch &=
(esperColumn.equals(sqlResult.getDate(attributeList[j].replaceAll(" ", ""))));
                                                System.out.print(attributeList[j] + "=" +
esperColumn + "," + sqlResult.getDate(attributeList[j]) + " ");
                                        } else {
                                                rowMatch &=
(esperColumn.equals(sqlResult.getString(j+1)));
                                                System.out.print(attributeList[j] + "=" +
esperColumn + "," + sqlResult.getString(attributeList[j].replaceAll(" ", "")) + " ");
                                        }
                                }
                        }
                        System.out.print("| Matching=" + rowMatch + "\n");
                        resultMatch &= rowMatch;
                        i++;
                }
                System.out.println("Results match = " + resultMatch);
                return resultMatch;
        }

        /**
         * Get all data in SQL database and convert them into Esper Events
         * @param db
         * @param conn
         * @param cepRT
         * @throws SQLException
         * @throws InterruptedException
         */
        public void getEventsFromDB(String db, Connection conn, EPRuntime cepRT) throws
SQLException, InterruptedException {
                String query = "SELECT * FROM " + db;
                ResultSet rs = DatabaseGenerator.executeStatement(conn, query);

                while (rs.next()) {
                        acimqp1 event1 = new acimqp1(rs.getInt("EVENTID"),
rs.getString("VSTRING1"), rs.getFloat("VFLOAT1"),
                                        rs.getDate("VDATE1"), rs.getInt("VINT1"),
rs.getInt("VINT2"));
                        System.out.println("Sending event: " + event1);
                        cepRT.sendEvent(event1);
                }
        }
```

```
	/**
	 * Execute Esper query against 1 event stream
	 * @param esper
	 * @throws SQLException
	 * @throws InterruptedException
	 */
	public void runEsper1Event(EsperGenerator esper) throws SQLException,
InterruptedException {
		esper.Print();
		//The Configuration is meant only as an initialization-time object.
		Configuration cepConfig = new Configuration();
		cepConfig.addEventType("acimqp1", acimqp1.class.getName());
		EPServiceProvider cep =
EPServiceProviderManager.getProvider("myCEPEngine", cepConfig);
		EPRuntime cepRT = cep.getEPRuntime();
		EPAdministrator cepAdm = cep.getEPAdministrator();

		EPStatement cepStatement1 = cepAdm.createEPL(esper.getEsper());
		cepStatement1.addListener(new CEPListener());

		getEventsFromDB("ACIMQP1", this.connection, cepRT);
		cepStatement1.removeAllListeners();
	}

	/**
	 * Execute Esper query against 1 event stream
	 * @param esper
	 * @throws SQLException
	 * @throws InterruptedException
	 */
	public void runEsper1Event(String esper) throws SQLException, InterruptedException {
		System.out.println(esper);
		//The Configuration is meant only as an initialization-time object.
		Configuration cepConfig = new Configuration();
		cepConfig.addEventType("acimqp1", acimqp1.class.getName());
		EPServiceProvider cep =
EPServiceProviderManager.getProvider("myCEPEngine", cepConfig);
		EPRuntime cepRT = cep.getEPRuntime();
		EPAdministrator cepAdm = cep.getEPAdministrator();

		EPStatement cepStatement1 = cepAdm.createEPL(esper);
		cepStatement1.addListener(new CEPListener());

		getEventsFromDB("ACIMQP1", this.connection, cepRT);
		cepStatement1.removeAllListeners();
	}
}
```

TestTesting5(with the output of Testing5):

```
public class TestTesting5 {

	@Test
	public void test() throws SQLException, InterruptedException {
		String sql = "SELECT COUNT(*) FROM ACIMQP1"
					+ "\nWHERE ACIMQP1.vInt2 < 1000"
					+ "\nGROUP BY vInt2";
		String[] attributesList = {"count(*)"};
		Connection conn = DatabaseGenerator.connectDatabase();
		EventSetup ev = new EventSetup(conn, sql, attributesList);
		ev.runEsper1Event(Testing5.genEsper());
		assertTrue(ev.compareResults());
		Thread.sleep(1000);
	}
}
```

Output:

```
SELECT COUNT(*) FROM acimqp1.win:keepall()
	WHERE acimqp1.vInt2 < 1000
	AND acimqp1.vFloat1 < 50.0
	GROUP BY vInt2
Sending event: EventID: 1 vstring1: US vfloat1: 0.2 vdate1: 2014-02-02 vint1: 10 vint2: 100
Event received: {count(*)=1}
Sending event: EventID: 2 vstring1: US vfloat1: 0.4 vdate1: 2013-03-03 vint1: 20 vint2: 200
Event received: {count(*)=1}
Sending event: EventID: 3 vstring1: US vfloat1: 0.6 vdate1: 2012-04-04 vint1: 30 vint2: 300
Event received: {count(*)=1}
Sending event: EventID: 4 vstring1: US vfloat1: 0.8 vdate1: 2011-05-05 vint1: 40 vint2: 400
Event received: {count(*)=1}
Sending event: EventID: 5 vstring1: US vfloat1: 1.0 vdate1: 2010-06-06 vint1: 50 vint2: 500
Event received: {count(*)=1}
Sending event: EventID: 6 vstring1: US vfloat1: 1.2 vdate1: 2009-07-07 vint1: 60 vint2: 600
Event received: {count(*)=1}
Sending event: EventID: 7 vstring1: US vfloat1: 1.4 vdate1: 2008-08-08 vint1: 70 vint2: 700
Event received: {count(*)=1}
Sending event: EventID: 8 vstring1: US vfloat1: 1.6 vdate1: 2007-09-09 vint1: 80 vint2: 800
Event received: {count(*)=1}
Sending event: EventID: 9 vstring1: US vfloat1: 1.8 vdate1: 2006-10-10 vint1: 90 vint2: 900
```

```
Event received: {count(*)=1}
Sending event: EventID: 10 vstring1: US vfloat1: 2.0 vdate1: 2005-11-11 vint1: 100 vint2: 1000
Sending event: EventID: 11 vstring1: US vfloat1: 2.2 vdate1: 2004-12-12 vint1: 110 vint2: 1100
Sending event: EventID: 12 vstring1: US vfloat1: 2.4 vdate1: 2003-01-13 vint1: 120 vint2: 1200
Sending event: EventID: 13 vstring1: US vfloat1: 2.6 vdate1: 2015-02-14 vint1: 130 vint2: 1300
Sending event: EventID: 14 vstring1: US vfloat1: 2.8 vdate1: 2014-03-15 vint1: 140 vint2: 1400
Sending event: EventID: 15 vstring1: US vfloat1: 3.0 vdate1: 2013-04-16 vint1: 150 vint2: 1500
...
COMPARE RESULTS:
Esper vs SQL
count(*)=1,1 | Matching=true
| Matching=true
```

TestTesting6(with the output from Testing6):

```
public class TestTesting6 {

	@Test
	public void test() throws SQLException, InterruptedException {
		String sql = "SELECT COUNT(*) FROM ACIMQP1"
				+ "\n\tWHERE ACIMQP1.VINT1 < 1000"
				+ "\n\tORDER BY VINT1 ASC ";
		String[] attributesList = {"count(*)"};
		Connection conn = DatabaseGenerator.connectDatabase();
		EventSetup ev = new EventSetup(conn, sql, attributesList);
		ev.runEsper1Event(Testing6.genEsper());
		assertTrue(ev.compareResults());
	}
}
```

Output:

```
SELECT COUNT(*) FROM acimqp1.win:keepall()
	WHERE acimqp1.vInt1 < 1000
	ORDER BY vInt1 ASC
Sending event: EventID: 1 vstring1: US vfloat1: 0.2 vdate1: 2014-02-02 vint1: 10 vint2: 100
Event received: {count(*)=1}
Sending event: EventID: 2 vstring1: US vfloat1: 0.4 vdate1: 2013-03-03 vint1: 20 vint2: 200
Event received: {count(*)=2}
Sending event: EventID: 3 vstring1: US vfloat1: 0.6 vdate1: 2012-04-04 vint1: 30 vint2: 300
Event received: {count(*)=3}
Sending event: EventID: 4 vstring1: US vfloat1: 0.8 vdate1: 2011-05-05 vint1: 40 vint2: 400
Event received: {count(*)=4}
```

```
Sending event: EventID: 5 vstring1: US vfloat1: 1.0 vdate1: 2010-06-06 vint1: 50 vint2: 500
Event received: {count(*)=5}
…
COMPARE RESULTS:
Esper vs SQL
count(*)=99,99 | Matching=true
| Matching=true
Results match = true
```

TestTesting7(with the output from Testing7):

```
public class TestTesting7 {

	@Test
	public void test() throws SQLException, InterruptedException {
		String sql = "SELECT COUNT("
				+ "\nCASE 1"
				+ "\nWHEN 1 THEN 10"
				+ "\nWHEN 2 THEN 20"
				+ "\nELSE 999"
				+ "\nEND"
				+ "\n) FROM ACIMQP1"
				+ "\nWHERE VINT1 < 1000"
				+ "\nORDER BY VINT1 ASC";

		String[] attributesList = {"count(case 1 when 1 then 10 when 2 then 20 else 999
end)"};
		Connection conn = DatabaseGenerator.connectDatabase();
		EventSetup ev = new EventSetup(conn, sql, attributesList);
		ev.runEsper1Event(Testing7.genEsper());
		assertTrue(ev.compareResults());
	}
}
```

Output:

```
SELECT COUNT(
	CASE 1
		WHEN 1 THEN 10
		WHEN 2 THEN 20
		ELSE 999
	END
	) FROM acimqp1.win:keepall().win:time(1 sec)
	WHERE acimqp1.vInt1 < 1000
	ORDER BY vInt1 ASC
```

```
Sending event: EventID: 1 vstring1: US vfloat1: 0.2 vdate1: 2014-02-02 vint1: 10 vint2: 100
Event received: {count(case 1 when 1 then 10 when 2 then 20 else 999 end)=1}
Sending event: EventID: 2 vstring1: US vfloat1: 0.4 vdate1: 2013-03-03 vint1: 20 vint2: 200
Event received: {count(case 1 when 1 then 10 when 2 then 20 else 999 end)=2}
Sending event: EventID: 3 vstring1: US vfloat1: 0.6 vdate1: 2012-04-04 vint1: 30 vint2: 300
Event received: {count(case 1 when 1 then 10 when 2 then 20 else 999 end)=3}
Sending event: EventID: 4 vstring1: US vfloat1: 0.8 vdate1: 2011-05-05 vint1: 40 vint2: 400
Event received: {count(case 1 when 1 then 10 when 2 then 20 else 999 end)=4}
Sending event: EventID: 5 vstring1: US vfloat1: 1.0 vdate1: 2010-06-06 vint1: 50 vint2: 500
Event received: {count(case 1 when 1 then 10 when 2 then 20 else 999 end)=5}
…
COMPARE RESULTS:
Esper vs SQL
count(case 1 when 1 then 10 when 2 then 20 else 999 end)=99,99 | Matching=true
Results match = true
```