INSIGHT Expression Language

Geoff Howe

July 1, 2013

# Introduction

What follows is a study of the kinds of expressions needed to be supported for rules and conceptual-level logic templates to support the automated process assembly features of the Insight platform. Expressions are written in an abstract manner, irrespective of a specific instantiation allowing a purely logical statement of some domain condition or rule. Operationally, the conceptual objects, attributes and relationships specified are placed into particular representational contexts through a mapping process which ties them to physical references. This association to a physical, representational context we’ll call an “interpretation” after the first order logic concept which it approximates. The combination of the conceptual-level logic statement, the mapping and the physical representation meta data allow the Insight platform to automatically generate and execute Java code to perform various algorithms which are logically the same, but physically different in their particulars.

A more thorough explanation of this capability must be found elsewhere.

Typical expressions are a mainstay of programming paradigms. Comparing data objects in various combinations and times, coupled with branching logic is the essence of programming. This document will cover only the abstract nature of the expressions themselves, and will let other documents describe how these expressions might be combined with logic. Examples intended of the combination of the two include a rules engine, and a data query/selection engine, along with other uses.

Taken in the abstract, expressions can be interpreted as logic statements within a domain consisting of a “universe of discourse”. Multiple “domains” in this sense exist simultaneously, so any coherent set of expressions are written within a context in which they are true or false, implicitly. Each domain of discourse may present different expressions, which may appear to contradict each other when looked across domains (e.g., in domain A, P is true, but in domain B, P is false). This is to be expected, and so our expression language must be context-bound, with only predicates within the same domain being allowed to interact. We will worry about this detail later. For the bulk of this discussion, we will assume all examples are written within the same contextualizing domain, unless explicitly mentioned otherwise.

# Conceptual Expressions

## Basic Components

Expressions will be written by referring to concepts in an Entity Relationship model. Entities will be referenced by name, Entity Relationships as well. Both will permit reference of Attributes and “Named Conditions” associated with them, either implicitly or explicitly. All statements written in terms of these types will be conceptual in nature, and shall not imply or impose any particular data structure or representation. Hence, the expressions that are written using these types will simply be logical statements.

In addition, while the language implies the presence of an Entity Relationship model, we will not require that such a model exist a priori of writing the expressions. Rather, we will permit the introduction of Entity Relationship elements through their discovery within the syntax and placement of terms in the various expressions which are written. While this could lead in the short run to duplicative or contradictory references, we feel that the importance of the free-flowing logic is more important than a rigid conformance. A key concept of the INSIGHT platform is the alias. Hence calling something different names in two parts of a context should not be considered a problem. [[1]](#footnote-1)

Every Entity shall have an Attribute named [ID] which refers to the concept of a unique identifier of the Entity. Without specifying a structure for the [ID], the intent is to abstract out the notion of uniqueness of an entity from the representation of the entity. In actuality, the conceptual attribute [ID] is intended to stand in for any and all possible representations and instantiations of identifiers. Hence, any possible representation of an identifier will ultimately be accepted as a valid instantiation if it satisfies the uniqueness condition on a set of actual representations in the same context. This will become clearer in the section on Mapping.

Entity Relationships shall also have the notion of uniqueness, however, this will be defined by the accumulation of identifiers from other Entities. For example, a person may live in many homes over the course of their lives, and sometimes a person may have multiple homes at the same time.

Thus the relationship between a PERSON entity and a HOME entity could be many to many. We can define an Entity Relationship PERSON-HOME by defining its [ID] to be constructed from the combination of the PERSON.[ID] and the HOME.[ID]. The relationship could then also have its own Attributes, such as [START DATE] (when the person first started using the home).[[2]](#footnote-2)

NOTE: Entity Relationships have several standard attributes, such as [CARDINALITY] and [RELATIONSHIP TYPE], but the exact meaning of these will be dependent on the context which implements them.

The Attributes of an Entity or Entity Relationship are characteristics or observed facts about objects of these types. Anything can be an Attribute. For the PERSON entity we might have: [FIRST NAME] and [GENDER]. For HOME we might have [STREET ADDRESS], [CITY] and [TYPE OF STRUCTURE]. For PERSON-HOME we might have [PREFERENCE RANK], [PRIMARY OCCUPIED SEASON], and the aforementioned [START DATE]. Attributes are always defined with respect to an Entity or Entity Relationship.

The “Named Condition” is a special variant form of Attribute representing in strict sense a Boolean value. These are stand-ins for possibly extremely complex conditions, hence their name. Like Attributes, the Named Conditions are defined with respect to an Entity or Entity Relationship. Their interpretation should always be of a true or false condition describing the thing they are attached to. For example, the PERSON entity might have a Named Condition of <ADULT> indicating whether the Person represented is an “adult” or not an “adult”. Only attributes which are truly binary in nature should be considered as Named Conditions.

## Context

Context is in general quite a complex, nuanced concept, with many layers and implications. There are several layers to Context within the INSIGHT universe. At the level of the Expression Language described here, Context is defined as the universe of discourse within which the logical statements are true and consistent. This definition is purposefully general to permit a wide variety of self-consistent “universes” to be created. Within one Context, a single name of an Entity, Entity Relationship, Attribute or Named Condition will always reference the same concept. Across two contexts, the same lexical name should be assumed to mean different things, albeit with very subtle differences, depending on the circumstances.

There are additional definitions and constraints on Context at different levels within INSIGHT, but these will be discussed elsewhere.

Knowing the Context of an Expression Statement, therefore, will be critical to applying it. An Expression defined in one Context most likely will not be true in another one, nor interpretable, nor consistent with expressions defined in other contexts. Hence, every expression must somehow indicate the Context within which it has been defined. For INSIGHT, Contexts will initially correspond roughly with specific applications. Over time, multiple applications may be built, all sharing the same Context.

From a notational aspect, Context can be implied by the collection of Expressions themselves that appear together. This collection may itself be named, and may appear in the Expression listing, or may be derived by any systems or processes which are written explicitly to assume the Context. For now, our examples will assume that a context exists for every expression we inspect. Later, we’ll show an example of defining a Context explicitly within the text describing a set of Expressions.

## Reference Notation

In order to refer to Entities, Entity Relationships, Attributes and Named Conditions, a few simple conventions have been defined, as follows. First, all of these are named using upper case terms and phrases. Within one context, the same name represents the same idea.

Entities and Entity Relationships will be referred to by name within square brackets with no other embellishments or qualifiers. Names must always be capitalized. Operators and language key words should always be in lower case. Examples:

1. [PERSON]
2. [HOME]
3. [PERSON HOME]

Note that by notation it is not possible to differentiate between Entities and Entity Relationships. This should not pose a problem at the level of expression intended, since both of these have only Attribute and Named Condition modifiers within the language. The only other difference is that Entity Relationships must explicitly define their own [ID]’s as a mixture of borrowed identifiers from other Entities and additional Attributes. References that have not defined their [ID]s in terms of the [ID]s of other entities will be assumed to be Entity references with the default [ID]. This convention will be covered later.

Attributes can appear in direct context with their parent Entity or Entity Relationship, or in certain circumstances in a simplified form. The attribute names also appear in square brackets, but must always be preceded with a period. Thus the following examples:

1. [PERSON].[FIRST NAME]
2. .[GENDER]
3. [HOME].[STREET ADDRESS]
4. .[CITY]

Named Conditions also appear in context, and also in upper case letters and full words. They are, however, surrounded by angle brackets, as the following examples show.

1. [PERSON].<ADULT>
2. .<ALIVE>
3. [HOME].<IN USE>
4. [PERSON HOME].<PRIMARY RESIDENCE>

Lastly, we define a Population to be a set of like Entities (or Entity Relationship instances). Within the INSIGHT Expression Language, a Population appears as a name in capital letters surrounded by curly brackets.

1. {UNDERGRADUATE STUDENTS}
2. {ACTIVE COURSES}

Populations can also have Attributes and Named Conditions defined against them. These will be depicted in the same fashion as when they appear as part of an Entity. Such Attributes are defined over the entire population, rather than over individual members of the Population. Hence, {POPULATION}.[COUNT] would be the number of Entity instances within the Population. Anonymous, or temporary Populations can also be defined within the context of a single expression, and then attributes over these sets can be applied. Such references will appear as a pair of curly brackets with nothing inside.

1. {}

## Conceptual Constants and Independent Values

Since the INSIGHT Expression Language exists at the conceptual level, certain types of expressions may be trickier to specify directly. A case in point would be the entire class of expressions where an Attribute is to be compared to some particular value. Since values are typically intrinsically tied to specific notations (i.e., representations), how to lexically reference a particular value in a language which refuses to specify a physical format for the value becomes a problem.

To solve this problem, the INSIGHT Expression Language introduces the notion of a “Conceptual Constant”. Obviously, this is still a notation and representation, but it will support the separation of the concept from the physical representation of any particular instance.

A Conceptual Constant is a reference to the MEANING of a representation. Examples could include:

1. #MALE#
2. #RED#
3. #JUNE 1 2013#

Each of these should be considered to represent the concept, and not any specific representation of the concept. Hence (1) represents the notion of “male-ness” and (3) represents the notion of the first day of June in 2013.

As the expression language is applied against an actual physical representation, in addition to tying Entities and Attributes to physical representations, these Conceptual Constants must also be mapped to specific representations. There will be more on this later in the document.

Please note, that just as there can be many representational structures used to hold Attribute values, there can be many representations which share the concept intended by the Conceptual Constant. Without much explanation, for example, #MALE# could be variously represented by various string, boolean and numeric values, such as “M”, 1, “MALE”, etc.

A related idea is that of the Independent Value. An Independent Value is a more general class of values, which includes all of the Conceptual Constants, plus a set of values which act more like Attributes in that they may take multiple values, but which are independent of any Entity or Entity Relationship frame. Examples of Independent Values include anything that can have different values, like the time of day, or a set of related but distinct values like ranges or collections.

1. ##TODAY#
2. ##NOW#
3. ##MEMBER ID#
4. ##EARLY TWENTIES#

Notationally, Independent Values are prefixed with double pound signs, while Conceptual Constants have only a single pound sign. The class of all built-in values within the INSIGHT engine are included in the set of Independent Values. Each of these represents the concept of a variable or changing value, and again, each of these may be physically represented in completely different and possibly conflicting ways within any given representational context.

Conceptual Constants and Independent Values should not be confused with Named Conditions, even though for some universe of discourse or domain, both may be closely linked. Named Conditions may be defined using expressions that include either Conceptual Constants or Independent Values. For example, the following might be an expression defining conceptually what the .<MALE> condition means in terms of a PERSON’s GENDER.

[PERSON].<MALE> given by [PERSON].[GENDER] equals #MALE#

However, other definitions and conditions may also be used to imply the truth value of .<MALE>.

## Basic Expression Notation

An Expression makes a logical statement. The statement may be true or false, and the resolution of an expression’s truth value may be used to set the direction of some software process. Named Conditions may be defined in terms of more detailed Expression Statements. Constraints, filters, triggers, if-then-else statements could all use Expressions.

Expressions consist of conditions which when evaluated produce true or false values. The following examples will illustrate many forms of Expression Statements:

1. [PERSON].[GENDER] equals #MALE#
2. [PERSON].[ID] equals [PERSON HOME].[PERSON].[ID]
3. [PERSON HOME].[START DATE] between #JUNE 1 2001# and ##TODAY#
4. [PERSON].[AGE] in ##EARLY TWENTIES#

Expressions can be combined into more complex statements using conjunction, disjunction and negation qualifiers. Parentheses may be used in the usual manner to group conditions which should be resolved before combining with other parts.

1. ([PERSON].[GENDER] equals #MALE# and [PERSON].<ADULT>)
2. ([PERSON].[FIRST NAME] equals [REALTOR].[FIRST NAME] and [PERSON].[AGE] in (#20#, #21#, #22#, #23#)) or ([PERSON].[LAST NAME] equals [REALTOR].[LAST NAME])
3. Not ([PERSON].[LAST NAME] equals [REALTOR].[FIRST NAME])

# Predicate Calculus

## Universal Quantifiers and References

The expressions used thus far represent unqualified statements. In the INSIGHT Expression Language, the default quantifier for every statement will be considered to be universal over the domain/context, unless existential quantification notations are added. For example, a statement such as the following would make a statement about a particular population of PERSONs in the context, due to an implied/assumed association to the set of all PERSONS. The assumption being that without explicitly stating it, that for every person in the set of all PERSONS in the domain, both the gender and adulthood status would be assessed in combination for each member.

1. ([PERSON].[GENDER] equals #MALE# and [PERSON].<ADULT>)

In particular, this condition might be used to define a population of PERSON Entities as being entirely adult males.

Universal quantification may be explicitly indicated as well using the “For … ()” syntax, as follows. The simplest, and most compact, notation may be used when all of the conditions follow from the same type of Entity or Entity Relationship. In these examples, the Entity indicated after the For term is the intended Entity whose Attributes and Named Conditions are then listed. Each of these can be interpreted as the definition of a set or population of instances of the particular Entity or Entity Relationship

1. For [STUDENT] (.<GRADUATED>)
2. For [DEGREE] (.<UNDERGRADUATE> and .[DEGREE TYPE] equals #BACHELOR OF SCIENCE#)

In the next set of examples, the quantification notation is used to define a localized prefix to provide short hand reference to specific instances within the parenthetical body. This notation is recommended for clarity, when additional quantifiers for different kinds of entities are also specified (examples to follow later).

1. For [STUDENT] S (S.<GRADUATED>)
2. For [DEGREE] D (D.<UNDERGRADUATE> and D.[DEGREE TYPE] equals #BACHELOR OF SCIENCE#)

## Existential Quantifiers

While universal quantification is powerful, it lacks the nuance necessary for a completely flexible specification. In particular, the statements without existential quantification are really no more powerful than propositional logic, and logical statements across the association of Entities are not really possible.

The INSIGHT Expression Language notation permits existential quantification to indicate a variable number of required instances of some conditioned Entity or Entity Relationship. Whereas first-order logic is generally not specific about the number of instances, INSIGHT requires an explicit statement. For example, if we wanted to define a population of STUDENTS who had completed two sections of a particular course, we could write the following statement.

1. For [STUDENT] S (At Least 2 [COURSE SECTION] CS using [ATTEMPTED COURSE] AC with AC.[ID] matches ([S.[ID], CS.[ID]) and AC.<PASSED> and CS.[COURSE ID] equals #PERFORMANCE 301#)

There are a lot of concepts depicted in this example.

1. The condition depicted constrains the entity STUDENT by testing conditions on two other related concepts (COURSE SECTION and ATTEMPTED COURSE).
2. The condition sets an explicit constraint (the existential quantifier) on the number of distinct COURSE SECTIONS which must be found in order to satisfy the condition.
3. The ATTEMPTED COURSE is an Entity Relationship, not an Entity, given the depiction of its [ID] function.
4. The COURSE SECTION [ID] is not necessarily the same as its Attribute named [COURSE ID]. While it is possible that [COURSE ID] forms part of the [ID], it is not necessary for us to know this in order to write the statement.
5. The ATTEMPTED COURSES Entity Relationship’s [ID] consists of at least STUDENT and COURSE SECTION [ID]s. It also consists of the Entity [TERM]’s [ID], which is not mentioned by the expression. Through omission, the expression is indicating that the [TERM].[ID] is unbound or free within the statement.
6. The “matches” keyword is an instruction to the INSIGHT expression evaluator to cycle through all instances of ATTEMPTED CREDITS, STUDENTS, and COURSE SECTIONS, and one-by-one match the named Attributes (hence the bound attributes) to the STUDENT and COURSE SECTION [ID]s. Only those combinations which actually exist within the ATTEMPTED CREDITS universal population will then be used to test the remaining clauses of the expression.
7. The result of the “matches” operator is a set of Predicates with bound and free variables corresponding to the components of an actual [ID]. The operator is used to prepare a resolution set to unify the clauses. Bound values of predicates must be equal, while free variables can match anything.

# Usage Patterns

## Named Condition Template Expressions

Named Conditions, as stated before, represent a single propositional statement of truth. But they can themselves be stated in more complex terms through the use of template expressions. Template expressions are simply logical formulas of arbitrary complexity.

In order to associate a template expression to a Named Condition, the two must be linked within the context of a specific Entity or Entity Relationship. The syntax for this is slightly different than for population definitions. The following is a simple example.

1. For [STUDENT] S { .<REGISTERED CURRENT TERM> given by (At least 1 [TERM] T using [ATTEMPTED COURSE] AC with AC.[ID] matches ([S.[ID], T.[ID]) and ##TODAY# between T.[START DATE] and T.[END DATE]) }

In this example we can see several conventions.

1. The set of Named Conditions are defined within a context (STUDENT).
2. The [TERM].[ID] is bound and the [COURSE SECTION].[ID] is free in this example.

More than one template expression may be defined for an entity, even more than one per Named Condition. This is accomplished by including more than one within the curly brackets and separating each by a semi colon. When the Named Condition is repeated with different templates, this should be interpreted as alternative logic. The correct interpretation would be of the inclusive or of the different expressions. Here is a trivial example of this syntax.

1. For [ENTITY] E { .<CONDITION ONE> given by .<SOME CONDITION>; .<CONDITION TWO> given by .<ANOTHER CONDITION>; .<CONDITION ONE> given by .<YET A THIRD CONDITION> }

This examples shows three Named Condition Template Expressions and two Named Conditions. CONDITION ONE can be satisfied by either .<SOME CONDITION> or ,<YET A THIRD CONDITION>. CONDITION TWO can be satisfied just by .<ANOTHER CONDITION>. Note also that all of these Named Conditions are associated with the Entity ENTITY.

## Population Definitions

Expressions shown previously are all anonymous. The Expression Language allows us to give names to each Population. A Population is a named and defined set of instances of some Entity, and a Relation Population is a named and defined set of instances of some Entity Relationship. Given a name, a Population’s definition is given as an expression in the INSIGHT Expression Language. The Population must be defined over the universal set of instances of the Entity. Populations are indicated in the notation by having names which are surrounded by curly brackets. The following are a few example Population definitions.

1. {UNDERGRADUATE STUDENTS}   
   @POPULATION [STUDENT]  
   @POP\_DEF .<UNDERGRADUATE> ()
2. {ACTIVE UNDERGRADUATE STUDENTS}   
   @POPULATION [STUDENT]  
   @POP\_DEF {UNDERGRADUATE STUDENTS} .<ACTIVE>
3. {NEW STUDENTS}  
   @POPULATION [STUDENT]  
   @POP\_DEF {FIRST TERM FRESHMAN} union {FIRST TERM TRANSFER STUDENTS}
4. {REGISTERED STUDENTS}  
   @POPULATION [STUDENT]   
   @POP\_DEF .<REGISTERED CURRENT TERM> given by (At least 1 [TERM] T using [ATTEMPTED COURSE] AC with AC.[ID] matches (.[ID], T.[ID]) and ##TODAY# between T.[START DATE] and T.[END DATE])

Number (2) shows that in addition to providing a means to define populations over Entities, the INSIGHT Expression Language also allows the author to define new Populations by adding further constraints to previously defined Populations. Properly considered, these would be “sub-populations”. Number (3) shows the union of two different populations. Number (4) shows an existentially quantified population.

For Population Definitions that present a condition template, in instantiating an interpretation, the template can always be ignored in lieu of a custom physical representation which satisfies the Named Condition in a manner different from what the template suggests. Hence in number (4) above, one interpretation would rely on mapping the template elements ([TERM] and [ATTEMPTED COURSE]) and another may simply set the .<REGISTERED CURRENT TERM> condition directly. This definition might be created in this way if the template is not needed independently of the population definition.

1. Eventually, tools for managing and visualizing the Entity Relationship Models implied by a set of expressions will be available. [↑](#footnote-ref-1)
2. Note that Entity Relationship [ID]s can be even more complex, to include additional Attributes as well as references to other Entity IDs. [↑](#footnote-ref-2)