Understanding Concepts and Connections in freeSCHEMA

At the heart of freeSCHEMA there are **Concepts** and **Connections**, which represent and structure data within the system.

1. Concepts

Concepts in freeSCHEMA are the fundamental building blocks, representing data objects like users, products, or documents. Each concept has the following key elements:

- **ID:** A unique identifier for the concept.
- **Category:** The broad group to which the concept belongs (e.g., a person, product, or document).
- **Type:** Specifies the kind of entity the concept represents (e.g., a customer, electronic product, or report).
- **Referent:** The actual data or value (e.g., a person's name, a product's model, or a document's title).

1.1 Create a Concept

Concepts in freeSCHEMA can be of two types: **Typological /Type Concept** and **Instance Concept**.

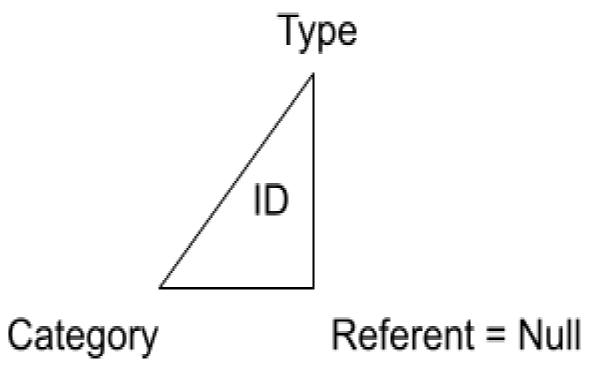
a) Typological /Type Concepts:

A **Type Concept** is a foundational concept that defines a category or class of entities in the system. In freeSCHEMA, type concepts are only created **once** and reused throughout the system. These concepts serve as templates for instance concepts. In freeSCHEMA, a type concept is visually represented by a **right-angled triangle** and always has the prefix "the_" (e.g., 'the_Person').

Key Points:

- **Category (left corner):** The category or class to which this type belongs.
- **Concept ID (center):** A unique identifier for the type concept in the system.
- **Subtype (top):** The specific name of the type, such as "Person" or "Product."
- **Referent (right corner):** Always null, as type concepts do not represent specific instances.

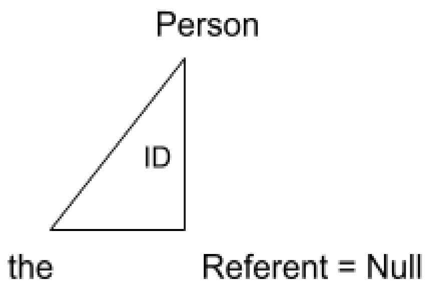
Visual Representation:



Code Example:

```
MakeTheTypeConceptLocal(

  typeString: string,     // The name of the type (e.g., 'Person')

  sessionId: number,      // The current session ID

  sessionUserId: number,  // The user ID who is creating or managing the session

  userId: number          // The user ID creating the type concept

);
```

Visual Representation:



- **Category (left corner)**: **the** — this type belongs to a general category called "the_."
- **Concept ID (center)**: this is the system's unique identifier for the type "the_Person."
- **Subtype (top)**: **the_Person** — the specific type being defined, which is "Person."
- **Referent (right corner)**: **null** — type concepts do not have a referent value.

Once created, this type concept can be used to generate multiple **instance concepts** like "John Doe" or "Jane Smith," both of which would be specific instances of **the_Person**.

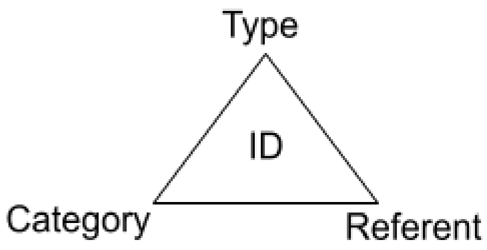
In freeSCHEMA, once a type concept is created, it remains part of the system and can be referenced whenever new instances of that type are created. This enables the system to maintain a structured and reusable data framework for managing various types of concepts across different contexts.

b) Instance Concept

An **Instance Concept** in freeSCHEMA represents a specific occurrence of a **Type Concept**, such as "Harry" for the **Type Concept "Person"**. It is visually depicted as an **upward-pointing triangle**.

Key Elements:

- **ID (center of the triangle):** The unique identifier for the instance concept within the system.
- **Concept Type (top of the triangle):** The type from which this instance concept is derived. For instance, "Person" in the case of "Harry".
- **Referent (right-hand side of the triangle):** The actual value represented by the instance concept (e.g., "Harry").



Code Example:

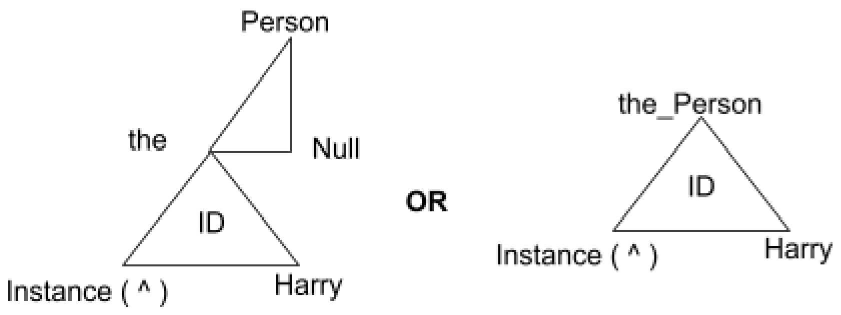
```
MakeTheInstanceConceptLocal(
  type: string,           // Type of concept (e.g., 'Person')
  referent: string,       // The specific instance (e.g., 'Harry')
  composition: boolean,   // Whether the concept is compositional or not
  userId: number,         // ID of the user creating the concept
  accessId: number        // Access level
);
```

Instance concept can also be of two types:

i) Instance Value Concept: An Instance Value Concept is unique and created only once for each combination of type and character value. For example, there can be only one instance of **Person** with the value "**Harry**", ensuring each instance is distinct within the system.

```
MakeTheInstanceConceptLocal(
    "Person",      // Type of concept (e.g., 'Person')
    "Harry",       // The specific value (e.g., 'Harry')
    false,         // Not compositional
    userId,        // User ID creating the instance
    4              // Access level
);
```

Visual Representation:

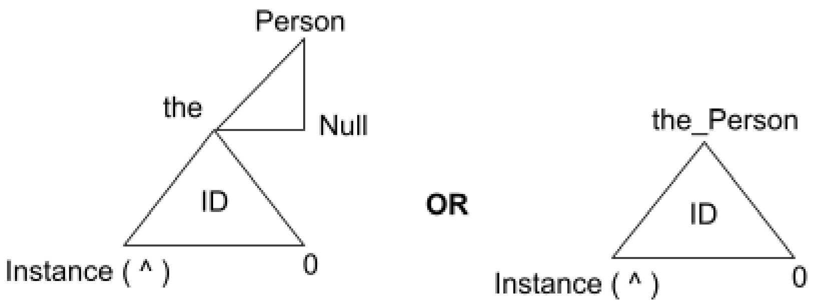


ii) Instance Compositional Concept:

An **Instance Compositional Concept** is a more complex structure that can contain other **value concepts**. It's similar to an object in **Object-Oriented Programming (OOP)**, where a single object can hold multiple attributes or other objects. Typically, these concepts have an empty character value ("") and may have a **referent ID** of 0, indicating that they serve as containers for other concepts. This is useful for representing more complex data structures, like a **"Person"** containing details such as **name**, **address**, and **email** each represented as separate concepts.

```
MakeTheInstanceConceptLocal(
    "Person",   // Type of concept (e.g., 'Person')
    "",          // No direct value (compositional)
    true,        // This is a compositional concept
    userId,      // User ID creating the instance
    4            // Access level
);
```

Visual Representation



2. Connections

Connections define relationships between concepts, enabling flexible and scalable data management. For example, a connection could describe the relationship between a "User" and a "Product" they purchased or between a "Person" and the "Company" they are employed by. Connections can be of two types: **Data Connections** and **Compositional Connections**.

Connections define relationships between concepts. These relationships allow for scalable data management. For example, a connection might define the relationship between a "User" and a "Product" they purchased.

Code Example:

```
CreateConnectionBetweenTwoConcepts(
    ofTheConcept: Concept,
    toTheConcept: Concept,
    linker: string,
    both?: boolean,
    count?: boolean
);
```

- **ofTheConcept:** The object representing the starting concept (e.g., a person).
- **toTheConcept:** The object representing the concept being connected (e.g., a product).
- **linker:** Describes the relationship (e.g., "Purchased", "EmployedBy").
- **both (Optional):** If set to true, creates a bidirectional connection.
- **count (Optional):** Tracks how many times this connection has occurred

2.1. Data Connections

A Data Connection is a straightforward relationship between two distinct concepts. For instance, if you have a concept for "John Doe" (as a person) and "Laptop" (as a product), a connection can be created to describe that "John Doe" purchased a "Laptop."

```
CreateConnectionBetweenTwoConcepts(
    personConcept,  // John Doe's concept object
    productConcept, // Laptop's concept object
    "Purchased"     // Describes the relationship
);
```

In this example:

a. **personConcept:** Represents the instance concept for "John Doe" (type: "the_Person").
b. **productConcept:** Represents the instance concept for "Laptop" (type: "the_Product").
c. **"Purchased":** This string describes the connection or relationship between these two concepts.

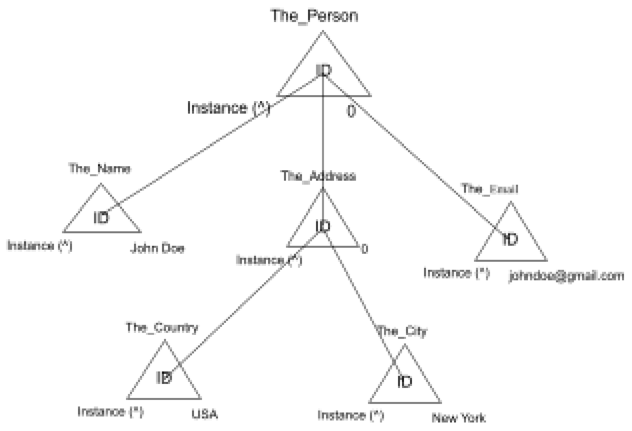
2.2 Compositions Connections

Compositional connections are used to define detailed, complex ideas by grouping related data into a single, cohesive structure. A composition is an instance of a type and is used to describe something like a person, object, or entity in detail. For example, the composition for a person could include their name, address, email and other defining properties.

```
let personComposition = {
    'Person' : {
     'Name': 'John Doe',
     'Address': {
        'Country': 'USA',
        'City': 'New York'
     },
     'Email' : 'johndoe@gmail.com'
   }
};
makeComposition(personComposition, userId);
```

In this case, the composition groups several related data points ‘Name’, ‘Address’, and ‘Email’ to describe the "Person" entity (John Doe) comprehensively. Compositional concepts are more complex and allow the representation of an entity that holds multiple properties under a single concept.

Visual Representation:


The_Person
ID
Instance (^)
0
The_Name
ID
Instance (^)
John Doe
The_Address
ID
Instance (^)
0
The_Email
ID
Instance (*)
johndoe@gmail.com
The_Country
ID
Instance (*)
USA
The_City
ID
Instance (*)
New York