多粒度时空对象数据库建模及导入工具设计说明（初稿）

# 1.需求分析

## 1.1工具设计目的

多粒度时空对象的管理和组织方式有两种：

1.基于多粒度时空对象标准交换格式的文件，以文件的方式组织和管理对象，这个工作由多粒度时空对象转换工具或者建模工具来完成。

2.将多粒度时空对象导入到时空数据库，用数据库来管理时空对象并对外提供数据访问的服务。

导入到时空数据库的数据源即是基于多粒度时空对象标准交换格式的对象文件。多粒度时空对象数据库导入工具就是专门用来将这些对象文件导入到时空数据库的。

## 1.2拟解决的问题

最首要的问题是多粒度时空对象数据库建模问题。具体就是以下几个点：

A.从文件中的数据到数据库中对应的记录，首先要解决的问题是ID编码问题。

交换格式是基于xml文件的，其中的大多数标签都没有ID属性。只有少数的对象具有ID属性：ObjectTemplate/Object/ObjectRelationClass/ObjectRelation等。数据库中的每一条记录都有主键（PK）,对于没有ID属性的标签，需要有统一的ID生成方法为其编码。但是由于一部分标签如Object本身就具有ID，所以保留其原始的ID。

B.有相当一部分数据是存在与其他的数据库中的，比如一些对象间的关联关系数据就是存在NEO4j这样的图数据库中的。这样的数据最好还是不要迁移到时空数据库中来，专门的数据库处理专门的数据既有效率又方便易用。这涉及到具体的这一部分数据表结构设计问题。

C.也一部分数据既非基础数据类型数据（char,int,float等）也非地理数据（矢量、栅格等空间数据），而是一些体量较大的二进制文件，比如建筑物对象的3d模型数据（.3damx）；这些文件可以入库管理，但是有可能导入和访问效率会有问题。如果能够不入库，数据库只是记录元信息，访问的时候将数据“引流”过去可能效果会好一些。

D.基于交换格式的对象，数据可能时分散在各个文件中的，而且对象间还有继承与派生的关系。数据入库前是要先将分散的数据合并起来还是维持原来分散的状态，待用户访问的时候再合并起来，这是两种不同的策略，需要权衡选择。

E.还有一些其他的问题下面会讲到。

# 二、设计分析

## 2.1 ID编码设计

考虑到数据量和编码的统一性和对原始ID的兼容性。ID设计如下：

表名称+记录导入日期（具体到分钟）+16位16进制编码数字

例如：ObjectTemplate-201807071200-ffffffffffffffff

对于如Object这样的带有原始ID的记录，用专门的字段Original\_ID来储存其原始ID。

主键和外键引用用统一编码的ID。

**2.2 数据库互操作**

对于存在于其他数据库的数据如图数据库等，需要有一种数据库间互操作的机制。多粒度时空对象数据库的数据库管理系统是基于PostgreSQl改造的，而PostgreSQL的一个扩展FDW（Foreign Data Wrappers）就可以一定程度上的实现PostgreSQl对其他的PostgreSQL或者MySQL等数据库进行互操作。对于不支持的操作功能或者数据库，FDW本身也提供了扩展机制，允许用户自定义FDW操作和数据库种类。

## 2.3 大体量二进制库外文件处理

上面提到，对有一些二进制文件不做入库处理，但是这些文件也不存在于其他的数据库之中。为了让用户可以像操作那些入库的数据一样操作这些数据，也需要一种访问机制。幸好，FDW默认自带的file\_fdw提供了对文件的读写操作。这个问题也迎刃而解。

## 2.4 合并策略

如果在入库前就对数据进行合并，那么后期当数据更新的时候，据需要连带更新大量的数据；举例来说:对象间有继承关系，Object继承自ObjectTemplate，当最上层的ObjectTemplate更新时所有继承自它的Object都要更新一遍，这个可能会影响数据库访问效率和性能。

如果保持对象文件的分散状态将数据入库，那么当更新上层对象所引起的连锁反应就会小很多；而当用户访问的时候再去合并能够保证数据的状态一致性和现势性。

综合考虑还是决定数据入库前不进行合并处理，访问时再进行合并操作。

## 2.4 null值处理

因为采用了入库前数据不合并的策略，ObjectTemplate中已经有的属性，其所派生的Object中就不会再出现，但是这些属性又是not null的，也就是说其必须出现在Object中，这就是一个矛盾。这里分两种情况处理：

1.这些属性是引用自数据字典（ST\_DataDictionary）中的项。这里可以在数据字典中添加一个空值项（NULL），当ObjectTemplate有值而没有合并到Object中时，Object的对应属性就填入“NULL”值。

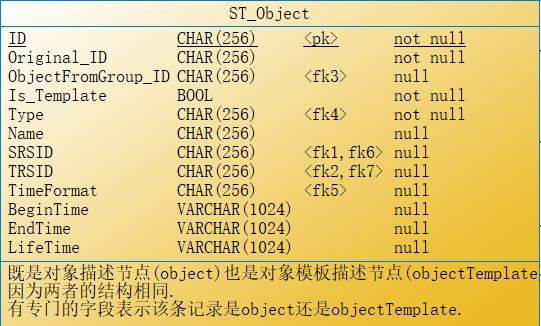
2.这些属性不是引用自数据字典中的项。这里也分两种情况：a.这个属性是引用自其他的数据表，这时候就在其引用的数据表中新添一个“NULL”项，属性引用这个项即可。b.这个属性不引用任何一张数据表，那么直接填入“NULL”即可。

注意：上面所说的“NULL”值不是null而是字符串字面值。

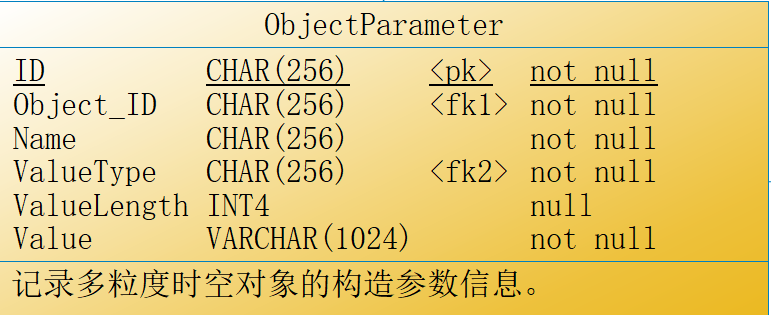
# 3.多粒度时空对象数据库表结构设计

按照中间交换格式中的多粒度时空对象的组成部分以及其之间的关联关系设计了如下若干表：

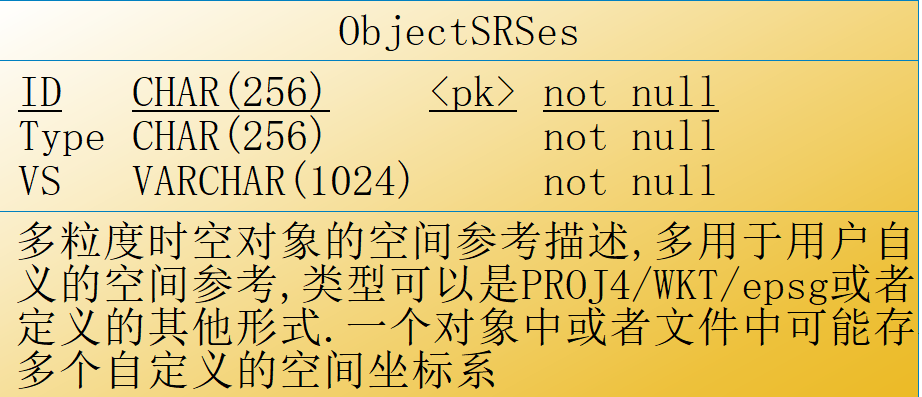
1.ST\_Object



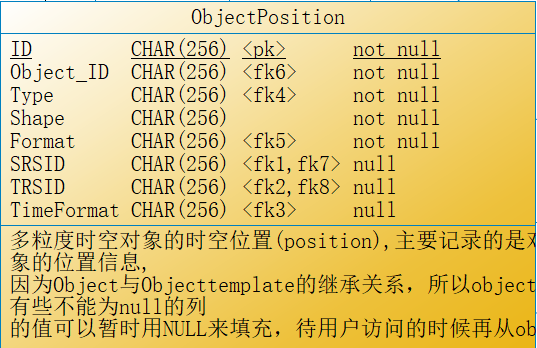
2.ObjectParameter



3.ObjectSRSes



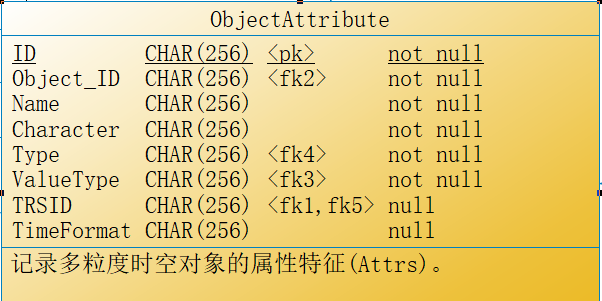
4.ObjectPosition



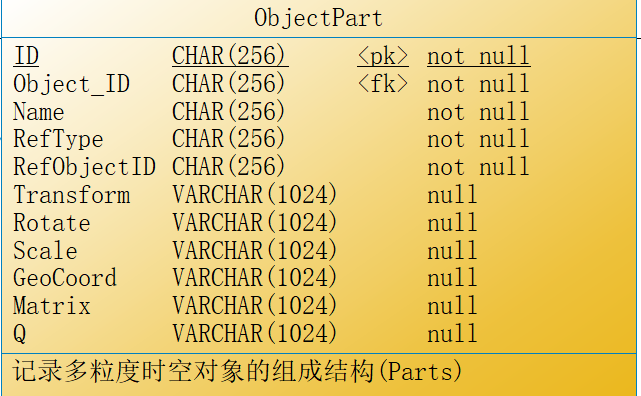
5.ObjectFrom



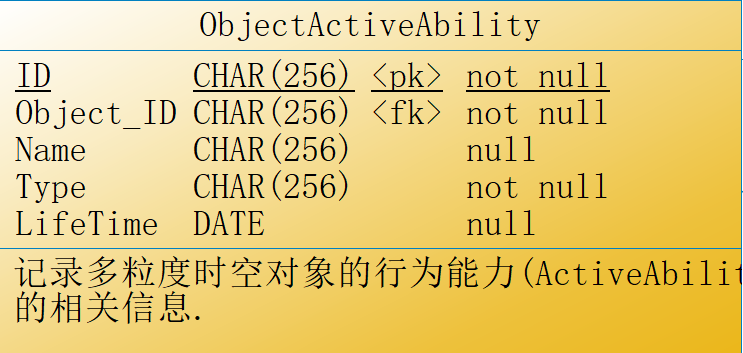
6.ObjectAttribute



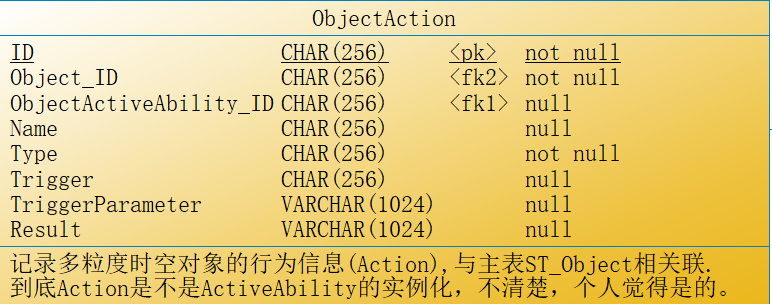
7.ObjectPart



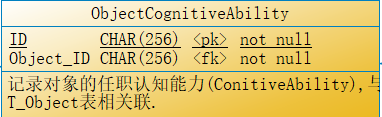
8.ObjectActiveAbility



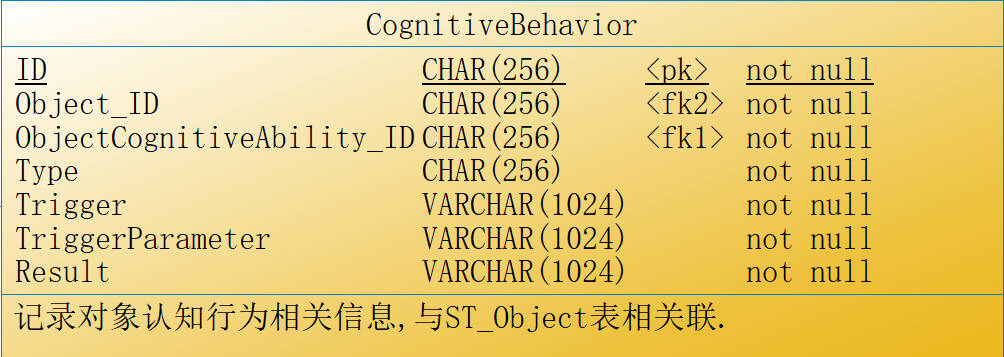
9.ObjectAction



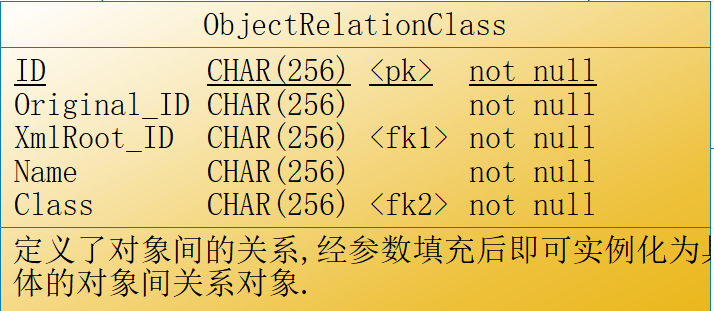
10.ObjectCognitiveAbility



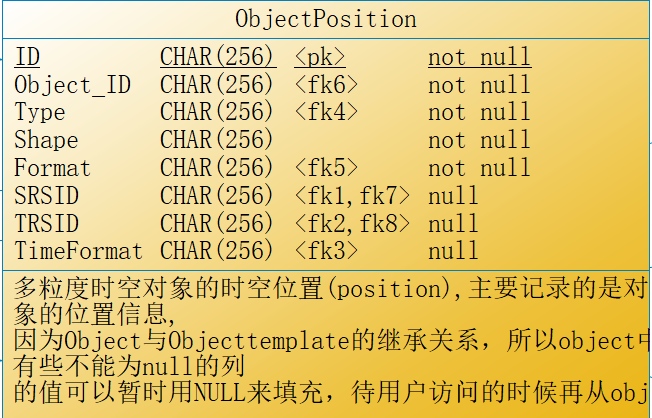
11.CognitiveBehavior



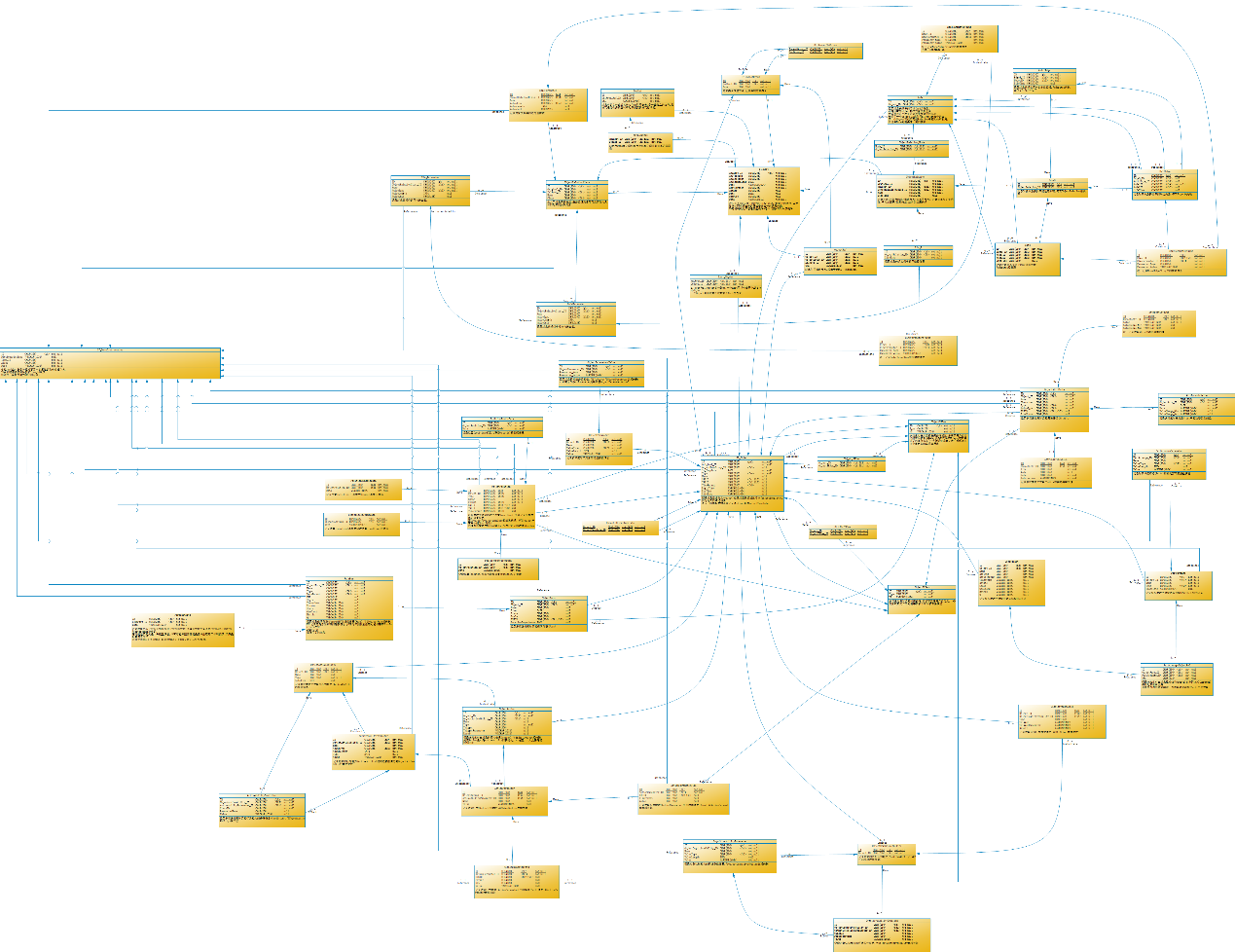
12.ObjectRelationClass



13.ObjectRelation



14.总体结构



以上是一些主要的组成部分的表结构设计，其他的一些附属表和关联表这里就不一一列举出来了。

# 4.多粒度时空对象入库流程设计

根据多粒度时空对象的各个组成部分的相关依赖关系，考虑到外键引用，各个组成部分的导入必须按照一定的顺序。其中ST\_DataDictionary因为之后的许多表要引用之，所以其必须最先导入，必须保证其中的数据项完备充足。

导入流程如下所示：



如上面刘晨所示：

1.首先将数据字典数据导入；

2.对对象文件统一ID编码并导入；

3.对文件间相互引用记录统一ID编码并导入；

4.对多粒度时空对象元信息记录统一ID编码并导入；

5.对多粒度时空对象的八元组的各个部分分别统一ID编码并导入；

6.对对象间关系数据记录统一ID编码并导入；

7.最终完成数据全部导入。