

# CS 8113D OBJECT ORIENTED DATABASE MODELS AND SYSTEMS

Fall 98 – Navathe

## Takehome Assignment # 1

### Question 1

#### ***Hull and King***

In terms of helping the reader to understand the semantic modeling area, this paper definitively does a great job (it's a review paper!). They compare and point the differences between semantic modeling and other modeling approaches such as object oriented or AI. They capture the most relevant components of semantic modeling under two different points of view: philosophical and practical. This clear distinction allows the reader to put in context the main branches of semantic modeling. At the same time a compact and simple notation is introduced along the paper. The notation is used to illustrate examples of the different concepts. This helps the reader to familiarize with their proposed semantics and finally understand them. The small number of constructs of their language makes it simple and powerful. Therefore, it is easy to concentrate on the main issues that the paper wants to address. The graphical notation significantly improves the understandability.

In section 3 of the paper (Survey) the authors include a review of the most prominent models under the different criteria they exposed in the previous sections. In addition they include two summary tables with the main schema components that every semantic model should address. This is an explicit proposal for a common framework to evaluate any data model. They go further: they actually include a full evaluation of several models.

In conclusion, I think that the paper tackles the evaluation criteria adequately. The following is a summary of the criteria they include for evaluating and analyzing the semantic modeling area:

#### *Philosophical Considerations for Semantic Models*

- Separation of conceptual and physical components.
- Decreased semantic overloading of relationship types.
- Abstraction Mechanism.

#### *Semantic Modeling "Checklist" proposed by Hull & King*

- Emphasis on attributes vs. emphasis on type constructors. Objects and Entities representations.
- Semantic Model Components.
  - Basic Building Blocks.
    - Atomic types. Distinction between abstract and printable.
    - Type constructors. Grouping and aggregation must be available.
  - Attributes: Single valued attributes, multiple valued attributes and *type* attributes.
  - ISA Relationships.
    - Overlapping vs. disjoint.
    - Covering and partition.
    - Subset vs. generalization.
- Usage of building blocks in schema construction:
  - Constraints on combining blocks.
  - Derived schema components. Specification of derivation rules, derived subtypes and derived attributes.
  - Static constraints.
    - Key dependencies.
    - Referential constraints.
    - Cardinalities on relationships.
    - Existence constraints.

- Data Manipulation Language.
  - Query capabilities (including abstract types).
  - Attribute referencing and manipulation.
  - Derived data manipulation.
  - Programming availability and language paradigm.

Still, there are improvements that can be done. First, their discussion on inheritance and multiple inheritance is not very rich. They do not provide a thorough discussion of the implications of inheritance and the problems and solutions available. Second, there is no way of determining which of the objects are weak or even distinguish between the different types of IS-A relationships using their symbols. The authors leave an “open” space for the designer to include these “constraints” to the side of the relationship, i.e., to write them down besides the arrow. Although this provides certain “modeling flexibility”, it will also make the overall representation very complex for large models. Third, they discuss many characteristics of many models. They review several models under different perspectives. But they do not provide a final characterization of the “real basic characteristics” that need to be supported. They do not compromise with a final set of characteristics, they just mention them all. Finally, if we look at the “language” capabilities they provide for derived data, the same criticism could be extended: they don’t give a set of functions or syntax that could be used as a base reference for model analysis.

## **SDM**

In terms of helping the reader to understand the components of a semantic model, this paper does a good job. They are more interested in presenting their model than discussing the area in general, but they still provide relevant discussions along the presentation. They first provide a high level view (criteria) to analyze contemporary database models. Afterwards, they introduce their semantic proposal with the philosophical motivations. A detailed discussion of the main components is included in the paper. This discussion could be used as a proposed “checklist” by the authors. Nevertheless, I consider this is not enough to understand the different views in the semantic modeling area. At the beginning of the paper they try to present the problematic of other models under their main criteria. But they are not always consistently evaluating other models using their specific constructs and solutions. In that sense, the reader cannot get the idea on why is this paper innovative or why its solutions are adequate. There is something extremely useful in this paper: a unique example is used to illustrate most of the semantic modeling capabilities. This is definitively a plus.

In conclusion, I think that the paper tackles the first evaluation criteria moderately and the second evaluation criteria adequately. The following is a summary of the criteria they include for evaluating and analyzing the semantic modeling area:

### *Philosophical Considerations for Semantic Models*

- Semantic expressiveness. Explicitly conveying meaning to the design.
- Relative views. Flexibility and logical redundancy.
- Abstract entity representation and relationships among entities. Existence.

### *Semantic Modeling “Checklist” implied by Hammer & McLeod*

- Relationship between classes and entities.
  - Class features: name, members, description, attributes, identifiers
  - Distinction between base class and non base class
  - Class vs. entity attributes
  - Class constraints
- Interclass connections. Multiple connections allowed.
  - Subclass connections.
    - Attribute defined subclasses.
    - User-controllable subclasses.
    - Set-operator defined subclasses.

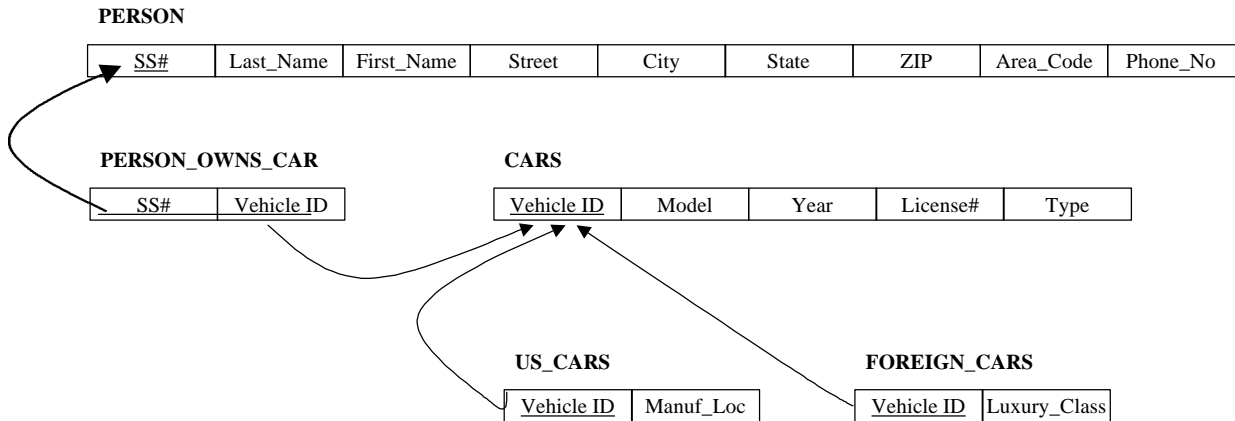
- Existence subclasses.
- Grouping connections.
  - Expression-defined
  - Enumerated
  - User-controllable
- Name classes.
- Attributes
  - Attribute features: name, value class, applicability, description, mandatory, not changeable
  - Single valued vs. multiple valued.
  - Non overlapping.
  - Schema defined attributes.
- Attribute interrelationships
  - Inverse
  - Matching
  - Derivation
  - Set operations
- Inheritance
- Other features
  - Mapping.
  - Ordering.
  - Recursive tracing.
  - Attribute predicates.
  - Duplicate and null value manipulation.

I can identify three main improvements to the paper. First, the authors should consider some graphical representation to guide the semantic modeling effort. Several models provide that capability and I think that it makes them very powerful to share among different disciplines. We cannot expect that a diagram can capture all the structure of their model, but it will certainly provide a good complement to their syntax. Second, there is no reference to any DML. They do not explore the implementation of state of the art querying to their modeling or modifications to the syntax to exploit their structure. Although this is not the purpose of the paper, a brief discussion could be useful. Finally, a simple table comparing their modeling capabilities with other semantic proposal will strengthen their position and give the reader a broader perspective of the area.

## Question 2 – RM/T

A)

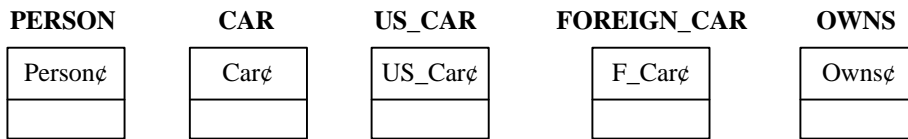
The following is the representation of the ER schema in the standard relational model:



B)

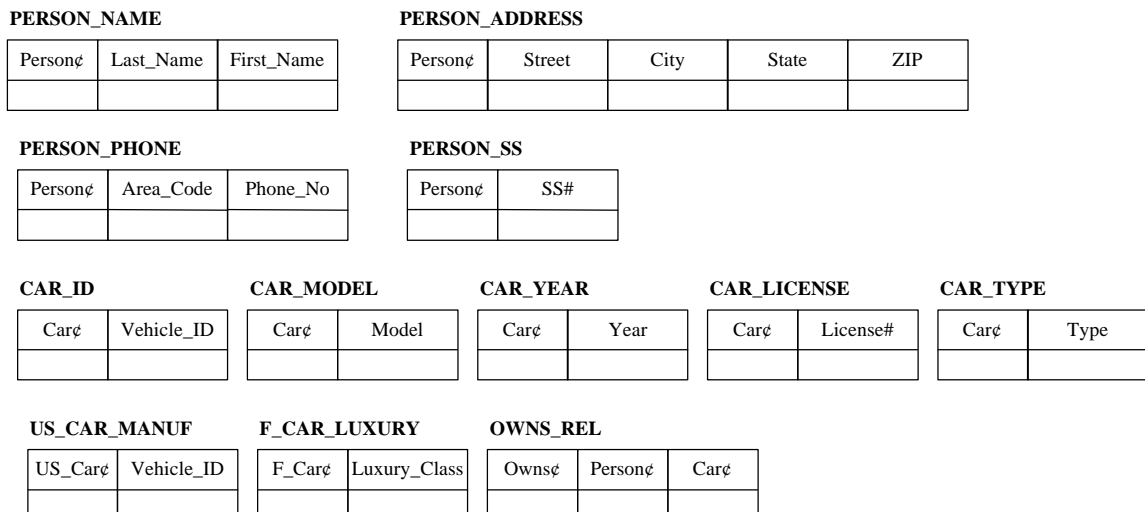
The same model will be represented as follows using the RM/T model:

*E- Relations*



- PERSON, CAR, US\_CAR and FOREIGN\_CAR are kernel entities. OWNS is an associative entity. There are no *characteristic* entities.
- In this particular implementation, the relation between PERSON and CAR was modeled as an entity association. I could have also modeled it as a non-entity association, in which case it will not be included in the E-relations.

*P- Relations*



## Property Graphs

### PG

<i>SUB</i>	<i>SUP</i>
PERSON_NAME	PERSON
PERSON_ADDRESS	PERSON
PERSON_PHONE	PERSON
PERSON_SS	PERSON
CAR_ID	CAR
CAR_MODEL	CAR
CAR_YEAR	CAR
CAR_LICENSE	CAR
CAR_TYPE	CAR
US_CAR_MANUF	US_CAR
F_CAR_LUXURY	FOREIGN_CAR
OWNS_REL	OWNS

### AG

<i>SUB</i>	<i>SUP</i>
PERSON	OWNS
CAR	OWNS

### UGI

<i>SUB</i>	<i>SUP</i>	<i>PER</i>
US_CAR	CAR	ORIGIN
FOREIGN_CAR	CAR	ORIGIN

In this example there are no CG, AGI, KG, US, AS, UP or AP relations, i.e., the relations are empty!

## C)

The following are the SQL queries on the RM/T representation of B).

### Query 1

List Foreign cars (Make, License#, Luxury-Class) owned by a person with SSN 1234:

```

EXEC SQL DECLARE Foreign_Cars CURSOR FOR
  SELECT Make, License#, Luxury-Class
  FROM ((PERSON_SS JOIN OWNS_REL) JOIN
        CAR_MODEL) JOIN CAR_LICENSE) JOIN F_CARS_LUXURY ON Carϕ=F_Carϕ
  WHERE PERSON_SS.SS# = 1234;

WHILE NOT (EOF) DO
  EXEC SQL FETCH Foreign_Cars
  INTO: amake, alicense, aluxury;
  PRINT (amake, alicense, aluxury);
ENDWHILE;

```

### Query 2

Is vehicle ID No. V123 a van?

```

EXEC SQL SELECT Type
  FROM CAR_TYPE JOIN CAR_ID
  WHERE CAR_ID.Vehicle_ID = "V123";
INTO atype;
IF atype= "van"
  PRINT ("The car V123 is a van");
ELSE
  PRINT ("The car V123 is not van");
ENDIF

```

*Query 3*

What subtypes exist for "CARS"?

```
EXEC SQL DECLARE subtypes_of_cars CURSOR FOR
  SELECT SUB
  FROM UGI
  WHERE SUP="CAR";

WHILE NOT ( EOF ) DO
  EXEC SQL FETCH subtypes_of_cars
  INTO: asub;
  PRINT (asub);
ENDWHILE;
```

*Query 4*

List the attributes of "PERSON".

```
EXEC SQL DECLARE attributes_of_person CURSOR FOR
  SELECT SUB
  FROM PG
  WHERE SUP="PERSON";

WHILE NOT ( EOF ) DO
  EXEC SQL FETCH subtypes_of_cars
  INTO: asub;
  Print (asub);
ENDWHILE;
```

### Question 3 - DAPLEX

Before answering the questions, and to avoid any confusion in the answers, I am including the DAPLEX definition of the schema:

```
DECLARE Person()=> ENTITY
DECLARE SS#(Person) => INTEGER
DECLARE Name(Person) => STRING
DECLARE Address(Person) => STRING
DECLARE Phone(Person) => STRING

DECLARE Car()=> ENTITY
DECLARE VehicleID(CAR) => STRING
DECLARE Model(CAR) => STRING
DECLARE Year(CAR) => INTEGER
DECLARE License#(CAR) => STRING
DECLARE Type(CAR) => STRING

DECLARE Owns(Person)>> Car
DECLARE US_Car()=> Car
DECLARE Foreign_Car()=> Car
DECLARE Manuf_Location(US_Car)> STRING
DECLARE Luxury_Class(Foreign_Car)> STRING
```

#### A)

The query “List Foreign cars (Make, License#, Luxury-Class) owned by a person with SSN 1234” will be written in DAPLEX as follows:

```
FOR EACH Person
  SUCH THAT SS#(Person)=1234
  FOR EACH x in
    INTERSECTION OF Foreign_Car, Owns(Person)
    PRINT Make(x)
    PRINT License#(x)
    PRINT Luxury_Class(x)
```

#### B)

##### *sports-car-ownership*

If the “sports-car-ownership” function I need to define is a BOOLEAN function that tests if a person owns a sports car, the DAPLEX definition is as follows:

```
DEFINE Sports-Car-Ownership(Person) =>
  COUNT(Owns(Person)SUCH THAT Type(Owns(Person))="sports")>0
```

If the function “sports-car-ownership” lists the sports cars of a person, then DAPLEX definition will be:

```
DEFINE Sports-Car-Ownership(Person) =>>
  Owns(Person)SUCH THAT Type(Owns(Person))="sports"
```

If the function sports-car-ownership list the persons that own a sports car, then the DAPLEX definition will be:

```
DEFINE Sports-Car-Ownership() =>> Person SUCH THAT
COUNT(Owns(Person)SUCH THAT Type(Owns(Person))="sports")>0
```

*three-car-ownership*

If the “three-car-ownership” tests if a person owns three cars (i.e. BOOLEAN function) the DAPLEX derived function will be:

```
DEFINE Three-Car-Ownership(Person) => COUNT(Owns(Person))=3
```

If the “three-car-ownership” gives the set of persons that own three cars, the DAPLEX definition will be:

```
DEFINE Three-Car-Ownership() =>> Person SUCH THAT COUNT(Owns(Person))=3
```

**C)**

The appropriate view of all the persons with no cars will be:

```
DEFINE persons-with-no-cars() =>> Person SUCH THAT
COUNT(Owns(Person))=0
```

**D)**

The following are additional features that can be included to the DAPLEX paper:

- ◆ *Key definitions.* DAPLEX does not explicitly deal with keys, and hence they must be user-defined. I will suggest to include a **DEFINE KEY** syntax that will define a key dependency. The advantage of this definition is that the system will know that you can use that function to refer to the entity itself (like an internal view) and that a constraint needs to be included in the system.
- ◆ *Ordered sets.* DAPLEX could provide the notion of an order output set for a function. The syntax =>> **XXXX ORDERED** will specify that the output set is ordered and hence functions as **SUCC( )** and **PRED( )** could be defined to move along the set.
- ◆ *Recursive tracing.* The possibility of recursively tracing all the entities involved in a cascade type of relationship could be an interesting feature to include.

## Question 4

I will organize the answer of this question as follows. I will first address the definition of keys of the papers. The “glossary” will be introduced along the discussion of each author. Whenever a glossary term appears along the discussion it is presented in bold font. After the discussion, I will explicitly answer the DAPLEX question posted in the exam.

### **Hull/King**

Due to the reviewing nature of this paper, several references to major historical definitions of keys are presented and sometimes mixed with the authors points of view.

The first<sup>1</sup> reference to a notion of *key/identifier* is presented on page 205 when referring to typical implementations of semantic data models: “(...) these abstract objects are referenced using internal identifiers that are not visible the user”. The authors include this statement when discussing what they call *abstract* data types, that is, entities in the model that map abstractions in the world. The authors argue that **internal identifiers** makes sense, because some types abstractions in the real world that correspond to different objects, might not have *printable* characteristics that uniquely identifies them! The **internal identifiers** correspond to an internal **object-id** concept, i.e., if an object exists in the nature then it has an existence independent of its characteristics that is captured by the system. Hence the user needs not to be aware of its value or manipulation. It is important to highlight that the authors mention that these internal identifiers correspond to objects in the *real world*. It seems that they will not agree on an internal **object-id** for an entity that does not exist in reality, but they are not very explicit in this matter.

On page 206, they introduce the **key** and **key dependency** concepts in the relational schema when “(...) the values occurring in one (or several) field(s) of a tuple determines the remaining field values of that tuple”. Note, that the definition of **key** is related with the *printable* attributes of the entities (field in the relational model): from the value(s) of the key(s) field(s), I am able to determine the values of the remaining fields. Observe that the **key** is itself a *printable* type that can be observed and modified by the user, in contrast with the **object-id** concept presented above.

On page 216, when discussing the *aggregation* of objects, the authors state that “(...) the *identity* of a grouping object is determined completely by that set [of objects]”. Note that in this definition, the aggregated object does not have an identity by itself, but the objects that compose it give its identity. This is an interesting annotation because they rule out the existence of an independent **object-id** for aggregated objects! Hence, this is a restriction when assigning **internal identifiers**. This statement has nothing to do with the **key** concept (that is related to *printable* characteristics). On page 220 they emphasize on the conceptual effects of this assumption when discussing the example of ENROLLMENT. They make a clear distinction of a **key** and the identity of an object (that might be handled using **internal identifiers**).

### **RM/T**

Codd is very clear when defining a **key** on the relational model: “ K is a *candidate key* (...) is a collection of attributes of R with the following time-independent properties: (1) No two rows of R have the same K-component, (2) if any attribute is dropped from K, the uniqueness property (1) is lost. For each base relation one candidate key is selected as the primary key”. I will only highlight two important characteristics on this definition:

- *Time independence*. The attributes that form the **key** ALWAYS satisfy the properties. The user can change the VALUES of the **key**, but the attributes that compose them always preserved both properties.

---

<sup>1</sup> They elaborate a little bit more on this matter in page 215.

- *Existence*: Every relation has a key, i.e., all of the attributes form a key.
- *User manipulation*. The user can change the values of the **key**, as long as the two properties are satisfied.

What he was ultimately looking for, was ways to refer to a tuple to extract information and relate it with other tuples in different relationships. He did not have in mind the *real* world he was trying to model. Observe that Hull and King use the same definition, except that they avoid (or forgot) to emphasize the time *independence* and the *existence* properties.

When extending the relational model the author tries to fix the problems that I mentioned above. He discusses the concept of **surrogate key**, i.e., a key that will “replace” the object within the model (page 409-411). The author considers that the **surrogate key** should be system-assigned and fully encapsulated in the entity: the user can never delete, nor refer or change its value. The value has no meaning in the application, is only a way of referring to the entity. The **surrogate key** is the way of representing the existence and uniqueness of the object in the real world. This will also be the way of representing the existence of any other entity in the model (even if it is composed by other entities with independent existence). This equivalent to the **object-id** concept and the **internal identifier** concept observed in Hull and King. The RM/T model heavily relies on the concept of **surrogate key/attribute** for the construction of the axioms, as illustrated by *Rule 3* in page 411.

As a final comment, Codd will not address the issue of **keys** any more after defining surrogates, because the identification of entities is completely resolved.

## **SDM**

Hammer and McLeod will introduce their point view when describing the criteria they consider essential in any database (pg. 353-354): “Moreover, the entities themselves must be distinguished from their syntactic identifiers (*names*); the user-level view of a database should be based on actual entities rather than on artificial entity names”. Their definition is very explicit. They consider that abstractions in the model should be handled internally and hidden from the user. Hence, the “existence” is handled by an **object-id** that is encapsulated in the application. We see the similarities between all the authors.

Latter in the paper (pg. 357), the authors discuss what they call *base classes*. In this discussion they introduce the notion of **identifier** of an object. The identifiers are the way of referring to the class members in a base class, i.e., is a set of attributes that can be used to refer to an entity. The authors explicitly say that **identifiers** provide a “one-to-one correspondence between the values of each identifying attribute or attribute group and the entities in the class”. This is the equivalent concept of a **key** in Codd or Hull and King papers. But this key is only for interaction with the user: internally the system refers to an entity with its **object-id**. It seems like the authors wanted to force the existence of a key in all *base classes*, but later on their discussion they explicitly mention that duplicates can be allowed in a *base class*. This means that entities of a *base class* could have all their attribute values identical, and hence no identifier can be defined!

To conclude this paper, I just want to emphasize that the only *base classes* are allowed to have **identifiers**. *Nonbase classes* do not have them! This is an important difference between the **key** definition of Codd and the **identifier** concept of SDM.

## **DAPLEX**

The author does not make explicit references to identification issues. But there are three phrases that give us an idea of his philosophical approach:

- On page 143 he says “To a large extent, DAPLEX is an attempt to provide such a framework for many ideas in this model [SDM]”.
- On page 144 he states “(...) [when a] function is applied, [an] entity itself is returned, not a (...) number or other identifier”.
- Finally, on page 145 he says, “ENTITY is the system-provided type of all entities”.

With these three insights, we are able to deduce his way of thinking: he is not concerned about the concept of **key** or **identifier**; that is up to the particular implementation. What he thinks is that entities have a system provided way of identification (**object-id**) that is not visible to the user, but that will drive the way that entities are handled. This is very clear along the discussion. The lack of interest in **keys** is particularly evident when he is showing a DAPLEX implementation of the relational model (page 158-159). He reduces the “uniqueness” property to a single value function without further discussion.

### **DAPLEX Question**

**Q:** How are keys of entity types and relationships types dealt with in the DAPLEX paper?

**A:** As I mentioned before, the author is not very concerned about the notion of key. DAPLEX does not have an explicit way of handling them. He is more concerned in providing a general data modeling capability. A key can be implemented as a user view, i.e., a function where the value of an attribute (e.g. a STRING or an INTEGER) get the actual entity for further manipulation. Therefore, DAPLEX allows the key definitions by combining *derived functions*, *single valued functions* and *user views*. For example, if we have a STUDENT entity and the Social Security number (SS#) is the “key”, then we can define a DAPLEX *derived function* to get the student name using its key attribute:

```
DECLARE Student()=>> ENTITY
DECLARE SS#(Student) => INTEGER
DECLARE Name(Student)=>STRING
DEFINE Student_SS() =>>SS#(Student())
DEFINE StudentName(Student_SS AS INTEGER)=>
    Name(THE Student(Student_SS))
```

Observe that the single arrow (single value) in the StudentName function represents the “uniqueness” property of the key. An additional constraint could be included to force unique SS numbers in the database.

Relationships in DAPLEX are functions on ENTITY types that map other ENTITY types. The cardinality of the relationship is handled by the type of function defined (single value or multiple value). Maximum and minimum cardinalities might be defined as constraints on the database. For example, suppose that the student can be registered in many courses (relationship between Students and Cours entity). But every student can not register more than six courses. The DAPLEX definition will be as follows:

```
DECLARE Student()=>> ENTITY
DECLARE Course()=>>ENTITY
DECLARE Registered(Student) =>> Course
DEFINE Are_Registered (Course) =>>
    INVERSE OF Registered(Student)
DEFINE CONSTRAINT max_register =>
    COUNT(Registered(Student()))<=6
```

## Question 5

This was a difficult question to address, especially because of the subjectivity of the concepts by themselves. Nevertheless, the following table summarizes my opinions:

Property	Hull/King	RM/T	SDM	DAPLEX
<i>Expressiveness</i>	Medium	Low	<b>High*</b>	Medium
<i>Readability</i>	High	Low	<b>High*</b>	Medium
<i>Understandability</i>	<b>High*</b>	Low	Medium	Medium
<i>Diagrammatic Notation</i>	<b>High*</b>	Low	Low	Low
<i>Formal Basis</i>	Low	<b>High*</b>	Low	Medium
<i>Minimality</i>	High	Medium	Medium	<b>High*</b>
<i>Correctness</i>	High	<b>High*</b>	High	High
<i>Completeness</i>	Low	Medium	<b>High*</b>	Medium
<i>Extensibility</i>	Medium	Medium	Medium	<b>High*</b>

### *Expressiveness & Readability*

SDM is the highest ranked model in these two properties. The authors had this in mind when developing their model: they want to provide a language that allows any users to get an idea of what the designer had in mind, without including too many symbols. Although they included several additional “terminology”, they also provided ways of transmitting meaning: rich associations, descriptive names and structured language and derivation.

### *Understandability & Diagrammatic Notation*

Although a SDM model might convey more meaning and even allow more readability, I think this does not make it necessarily more understandable. On the other hand the diagrammatic notation provided by Hull and King is definitively a key factor for making the model very understandable. Therefore, I ranked their paper as the highest in these two aspects. I also think that the diagrammatic notation of Hull and King makes it readable, although they cannot transmit as much expression as the SDM model. Finally, some diagrams are included in the DAPLEX paper, but unfortunately those are not considered a component of the language.

### *Formal Basis*

Codd’s RM/T is definitively a very formal model. Even the extension keeps the concise relational algebra properties with the advantage of allowing the user to include many other features of a semantic language. Its formal treatment of the null values reduces ambiguities. I consider important to highlight, that the DAPLEX language could also be very formal, although the authors decided not to include all of its formality in the paper.

### *Minimality*

This is a difficult property to determine. Clearly the Hull and King paper is minimal (almost as EER). Nevertheless, I consider that all the additional features included in DAPLEX are also needed in a SDM and in that sense this is for me a better level of *minimality*. It uses the simple concept of a function, defines internal “looping” variables, a basic set of rules and operators, and you are able to develop complex relationships and operations. It also gives enough insight to link it to an actual implementation or to another language.

### *Correctness*

I think that all of the models tackle the modeling area appropriately. They include most of the features needed. I decided to highlight Codd's paper because of its additional formal basis that provides an additional plus to its model. You can verify appropriate construction by checking the rules.

### *Completeness*

SDM gives the richest set of constructors to capture both simple and complex relationships. You can include some meaning in the design. It also provides language features to perform computations in the design phase. For this reason I believe that this is the more complete model.

### *Extensibility*

The way DAPLEX is conceived makes it very simple to implement and to incorporate additional features. The idea of reducing everything to a function is very powerful. It also leverages a multiplatform or even a cross-language implementation based on other models, as discussed in class.