DSE 250 Final – Fall 2017 December 7, 2017

Paul de Fusco
A07061018

For this final, you will fill in a “cheat sheet” of fundamental questions you need to consider when evaluating a novel (data model, query language) pair.

1 Models
1.1 What are the entities and the properties? At the very least, we expect every model to have a notion of entities/objects, with properties/attributes associated to them. The first thing to ask about a model is what notion corresponds to such entities, and what to their properties.

Relational Model - SQL:
Entities: tuples (a.k.a. “rows”) in tables
Properties: table column attributes

OO - OQL:
Entities/Objects:
Object: Atomic or Structured. Litteral: Atomic or Structured.
Object Characteristics: Identifier, Name, Lifetime, Structure
Object Main Attributes: hasname (bool), names, type

XML - XQuery:
This model very flexibly allows to represent relational and graph data by enforcing a hierarchical tree structure to the data. The basic components are element and text. Elements can contain attributes if specified either within the element’s tag or as special child nodes which from the pure implementation perspective it is nothing other than an another tagged entity within the hierarchical path. Tags merely function as labels for data values and entities gain relevance in terms of where they are contextually placed within the tree model. There is thus no referential integrity or notion of schema in the sense of the relational model. The first sections of the XML document contain metadata that can be used to specify element metadata.

neo4j graphs - Cypher:
The neo4j graph is a property graph: defined in graph theoretical terms as directed, vertex-labeled, edge-labeled multigraph with self-edges, where edges have their own identity. Vertices are defined as nodes and edges are defined as relationships. The following elements may exist:

- Entity: Node or Relationship
- Path
- Token: Label, Relationship Type, Property Key

graphs - gremlin:
The Gremlin model is characterized by the following entities:

- Graph: maintains a set of vertices and edges, and access to database functions such as transactions.
- Element: maintains a collection of properties and a string label denoting the element type.
    - ✦ Vertex: extends Element and maintains a set of incoming and outgoing edges.
    - ✦ Edge: extends Element and maintains an incoming and outgoing vertex.
- Property<V>: a string key associated with a V value.
    - ✦ VertexProperty<V>: a string key associated with a V value as well as a collection of Property<U> properties (**vertices only**)

JSON - N1QL (Couchbase's QL):

Json supports the following two data structures:

- Collection of name/value pairs
- Ordered list of values: includes array, list, vector or sequence

There is a loose notion of predefined object as there is a system JavaScript constructor that can be used to instantiate objects of a specific data structure. There are also data types used to store atomic values.

1.2 What notions of collections of entities exist? A model would be useless if it couldn't describe a collection of entities. Questions: what kinds of collections are supported? How do they relate to the standard programming language notions of record, set, multiset, list, array, map?

Relational Model - SQL:
The term collection can be used to describe a set of tuples (table entities). When a candidate key is enforced, the collection cannot contain duplicates and it becomes a bag.

OO - OQL:
The notion of collection takes the form of all collection objects inheriting from the Collection interface. Each collection object is further specialized into types which can take the values of set, bag, list, array, dictionary - which have logical equivalents in object oriented programming languages. Each collection type may provide additional interfaces enabling further relationships and operations within the collection dataset.
In standard OOP these are treated with class inheritance functionality.

XML - XQuery:
Because of the answer to 1.1, there is no real definition of collection other than what can be retrieved by navigating within a specific tree path to access elements and associated attributes.

neo4j graphs - Cypher:

The term ‘list’ is used to define a collection as a container that holds a number of elements which can have mixed types.

graphs - gremlin:
From the logical point of view, gremlin is a graph database with vertices and edges at the center of its model. From an implementation perspective, the Element class is used to model entities by maintaining a set of properties and a label for each. Vertex and Edge are additional classes extending the Element class.
Properties can be assigned to both vertices and edges. However, while edges can only be assigned a single key property, vertices can be associated with multiple properties of type ‘VertexProperty’.
A single Vertex property can have multiple values. Meta-properties can be used to assign properties on properties.

JSON - N1QL (Couchbase’s QL):
One can create objects by instantiating a data structure with the JavaScript constructor or by simply hardcoding objects within the JSON file structure and embedding data structures directly. A collection can be created by instantiating an array object, or creating an array of objects of different types (as we will see later complex values are allowed and so is nesting objects).

1.3 Is the model complex-valued? The model is complex-valued if the properties of an entity can in turn be collections of entities.

Relational Model - SQL:
If it were complex valued, it would be possible to have an entity nested within an entity i.e. having an entity as value in a given tuple property establishing a parent-child relation. This is not possible in the Relational Model as it would violate referential integrity.
However one could mimic the functionality by simply inserting multiple values within a tuple’s property value although this solution wouldn’t establish a physical parent-child relationship. Also, extensions to the Relational Model have integrated this functionality.

OO - OQL:
The model is complex valued because an object can be structured i.e. in turn contain additional objects if it takes the type of Immutable_Collection

XML - XQuery:
The model is complex valued in the sense that we can nest as many entities within other entities as desired. Logically we can think of these as collections. However, there is no particular pattern requirement for what can be nested within entities (except for the fact that it has to be one of the two allowed types - document or text), other than the fact that sub-entities will be characterized by the same tag-text association.

neo4j graphs - Cypher:
The model is not complex valued - a node cannot contain another node.

graphs - gremlin:
The model is not complex valued - a node cannot contain another node.

JSON - N1QL (Couchbase's QL):
The model is complex valued as any data structure can in turn nest other data structures.

1.3.1 Is the nesting of collections bounded? Can we have collections within collections etc. up to arbitrary nesting depth, or is this depth bounded?

Relational Model - SQL:
This model is not complex-valued so this question is N/A

OO - OQL:
The nesting of collections is bounded

XML - XQuery:
There is no bound to how many entities can be nested within entities.

neo4j graphs - Cypher:
N/A - nesting entities or collections not allowed

graphs - gremlin:
N/A - nesting entities or collections is not allowed

JSON - N1QL (Couchbase's QL):
The nesting of collections is un-bounded.

1.3.2 What notions of equality are defined? For atomic values (values of base type such as numeric, string, date, etc.) the notions of equality will likely be the standard ones adopted from classical programming languages. However, for complex values it makes sense to ask how they are compared as this is less standard. Possibilities are deep equality, which checks that two entities are equal by recursively checking that all their properties are pairwise equal. Related: what is "pairwise"? What are possible interpretations thereof?) Alternatively, shallow equality checks only that the non-collection properties of the two entities are pairwise equal.

Relational Model - SQL:
With respect to atomic values equality is a standard value based comparison. Predicate logic can be built to construct multiple equality conditions but the comparison is always made by single tuple property value.
This model is not complex-valued so the rest of the question is N/A

OO - OQL:

XML - XQuery:
There is no bound to how many entities can be nested within entities.

neo4j graphs - Cypher:
N/A - model does not allow for complex values

graphs - gremlin:
N/A - model does not allow for complex values

JSON - N1QL (Couchbase's QL):
There is no predefined key attribute to associate to a specified object in JSON. However, one can simply create an 'ID' field and assign it to the data structures used to model objects at will. However, Couchbase permits the use of primary keys to identify entities.

1.4 Does there exist a notion of identity? One thing to ask early on is whether entities have an identity, i.e. a designated property (subset of properties) whose value is unique across all entities.

Relational Model - SQL:
Each tuple can be uniquely identified by a candidate key i.e. a unique value assigned to each tuple in the relation. Assigning a candidate key is not mandatory; when one is not created, tuple duplicates are allowed. However, most implementations will greatly benefit from a primary key.

OO - OQL:
Yes - a notion of identity does exist as objects are characterized by an identifier.

XML - XQuery:
Nodes (elements) can be identified and referred to by their identity. Cycles and objects models can be described as well.

neo4j graphs - Cypher:
Yes - each node or relationship is defined with a unique identity. An entity (node or relationship) is assigned a set of properties, each of which are uniquely identified in the set by their respective property keys.

graphs - gremlin:
Yes - each node or edge is assigned a key.

JSON - N1QL (Couchbase's QL):
JavaScript does not provide object identifiers. However Couchbase permits to set and use primary keys for object identification.

1.5 If the model is complex-valued and identity-aware, are there corresponding equality notions? Can one compare entities by deep-value equality? By identity?

Relational Model - SQL:
N/A

OO - OQL:
yes - each object is the result of the instantiation of a class and implies the creation of a unique set of attribute values.

XML - XQuery:
One can compare entities by both identity and deep value equality.

neo4j graphs - Cypher:
N/A

graphs - gremlin:
N/A - model does not allow for complex values

JSON - N1QL (Couchbase's QL):
Deep value equality is allowed and can be performed from within the where clause, as long as primary keys have been set.

1.6 Is there a means to connect/link/relate entities? Can entity sets be related according to many-to-one, many-many, one-one relationships?

Relational Model - SQL:
Relations ("tables") can be linked with the implementation of referential integrity rules:

- One to One can be enforced by implementing a Primary Key on both tables, and placing a Foreign Key on the same table referencing the other table's corresponding attribute.
- One to Many can be enforced by implementing a PK in the table on the "one" side, and placing a FK on the "many" side referencing the corresponding attribute in the "one" side.
- Many to Many can be constructed with an "intermediary table" where the primary keys of both tables are referenced by foreign keys placed on the intermediary table.

OO - OQL:
Entities can be connected with referential integrity. This allows the traversal of objects within the model. All three relation types are allowed.

XML - XQuery:
Entities can be linked by establishing a tree hierarchical relationship between parent and child. Syntactically this implies simply creating an element nested within another one. This means one to one and one to many are the only relation types allowed.

neo4j graphs - Cypher:

Yes - nodes can be linked by establishing relationship to connect them with other nodes. One to One, One to Many and Many to Many can all be achieved simply by instantiating as as many relationships as wanted in between nodes.

graphs - gremlin:
Yes - vertices can be linked by means of edges. One to One, One to Many and Many to Many can all be achieved simply by instantiating as as many edges as wanted in between nodes.

JSON - N1QL (Couchbase's QL):
Entities cannot be linked in the same manner as of graph databases i.e. with relationship objects. However, the nesting of objects within objects in fact creates a parent-child relationship between the objects in many ways similar to XML's tree structure. So One to One and One to Many are both allowed but not Many to Many.

1.7 Is there a notion of type/kind/label associated to entities/properties? Start from your intuition gained from programming languages and see whether the standard notion of type has a counterpart in the model, or how it differs.

Relational Model - SQL:
A relation can be thought of as a class (i.e. "type") in an object oriented programming language i.e. a blueprint to instantiate entities belonging to it. In the relational model, tuples represent the objects instantiated from relations.

OO - OQL:
There is a notion of type as a type is an object instantiated from a class, just like in OOP languages. There is also the notion of label as each object can optionally be associated with a name.

XML - XQuery:
Typing is optional and done through definitions contained in the Document Type Definition (DTD).
Generally there are Elements and Attributes. They can be assigned a unique ID and labeled from within their associated XML tag. Advanced functionality allows to embed additional information within a tag.

neo4j graphs - Cypher:
Entities are either of type node or relationship. In addition, both are associated a unique identifier and can be labeled with a type (e.g. nodes can be of label 'Person' vs. 'Car'). However, there isn't a strict pattern constraint such as inheritance whereby if a node is of type 'Male' then it must be inheriting from type 'Person'.
Properties are also associated with a unique identifier and each node or relationship can be assigned as many as needed.

graphs - gremlin:
From a typing perspective, Edges and Vertices are subtypes of the Element type. Each of them is labeled by a key.

JSON - N1QL (Couchbase's QL):

As mentioned above, Json supports the following two data structures:

- Collection of name/value pairs
- Ordered list of values: includes array, list, vector or sequence

There is a loose notion of predefined object as there is a system JavaScript constructor that can be used to instantiate objects of a specific data structure. There are also data types used to store atomic values: Number, String, Boolean, Array, Value, Object, Whitespace, null.

1.8 Are all entities necessarily typed, or are types optional?

Relational Model - SQL:
By definition all tables are types, so types are not optional.

OO - OQL:
All entities are typed as they either take the form of atomic of structured.

XML - XQuery:
Type assignment is optional.

neo4j graphs - Cypher:
From an implementation perspective an entity must be typed either as a node or relationship, but each of them does not necessarily need to be assigned a label.

graphs - gremlin:
From an implementation perspective all entities must be typed as either Edge or Vertex.

JSON - N1QL (Couchbase's QL):
All entities will be typed in the sense that they will need to be instantiated either via the JavaScript constructor or through the use of one of the data structures defined above.

1.8.1 Can the same entity have more than one type? Or are types mutually exclusive?

Relational Model - SQL:
No - a tuple is the instantiation of a relation defined as a set of attribute values. A tuple of a different type would by definition represent a relation over a different attribute set, but in that case the type would be different. So Types are mutually exclusive.

OO - OQL:
No - multiple inheritance not allowed.

XML - XQuery:
Types are mutually exclusive.

neo4j graphs - Cypher:
It cannot - a node is either a node or a relationship. However, if we use the label as a way to implement a logical type e.g. 'Person' or 'Car' then it is possible to assign multiple labels to the same node e.g. one can both be labeled as 'Person' and 'Actor'.

graphs - gremlin:
It cannot - a node is either a Vertex or a Edge.

JSON - N1QL (Couchbase's QL):
Types are mutually exclusive.

1.8.2 Is there a subtyping relationship on types? A type t1 is a subtype of t2 if all entities of type t1 are implicitly also entities of subtype t2.

Relational Model - SQL:
A Primary Key - Foreign Key relationship between two relations' tuples is by definition a parent-child relationship. The PK-FK relationship enforces that at least one of the two table's values are uniquely mapped to the other table's values - if not both.

OO - OQL:
Yes - with the use of 'extends' one can allow a class to inherit from a parent class.

XML - XQuery:
Subtyping relationships are the essence of XML as elements are organized under a tree hierarchy. So yes.

neo4j graphs - Cypher:
No - there is no inheritance functionality

graphs - gremlin:
Both Edge and Vertex are subtypes of Element. Edge and Vertex cannot be extended further from a class inheritance perspective.

JSON - N1QL (Couchbase's QL):
The Object key value pair map cannot be subtyped. However, if one considers the logical data structures as types, then subtypes can be created by simply nesting one data structure into another.

1.8.3 Can a type be a subtype of several types that are not in direct/indirect subtyping relationships themselves?

Relational Model - SQL:
Yes - An implementation where multiple unlinked tables' attributes referenced the same attribute in an additional relation by means of FK would all "point" to the same child, thus making this a subtype of multiple types.

OO - OQL:
No - multiple inheritance not allowed.

XML - XQuery:
No - one element can only be contained by (or exist 'under') at most one element.

neo4j graphs - Cypher:
N/A - there is no inheritance functionality

graphs - gremlin:
No

JSON - N1QL (Couchbase's QL):
No - due to the tree hierarchy similar to XML one can type can only be nested within at most one other type.

1.9 How does the expressivity of this model compare to that of other reference ones? Model m is at least as expressive as model m0 if each value v 0 in m0 can be translated to a value v in m such that a faithful inverse translation from v to v 0 2 is possible.

Relational Model - SQL:
The relational model can express every relationship defined by other models, although complex graph model composed by nodes and edges are better modeled with systems such as graph databases because the edges are translated into relational tuples with difficulties related to referential integrity.

OO - OQL:
Very expressive - anything that can be modeled in SQL can also be modeled in OQL

XML - XQuery:
XML is the very expressive as its lack of referential integrity gives it superior flexibility and allows it to model any type of relationship or hierarchy. It is ideal for semistructured data where some structure is needed. It is possible to use it to model relational data although XML can do so with great redundancy and it is therefore not recommended for it. Graph data can also be modeled with ease in XML.
However this model does not allow for Many to Many relationships between tree nodes.

neo4j graphs - Cypher:
This model is very expressive when it comes to the ability to create relationships and the lack of constraints such as pattern inheritance or referential integrity. For the same reasons it is ideal for graph models but would model relational data poorly.
Graph traversal is made easy by the treatment of relationships as full entities with the same attribute and labeling/ID features as those given to nodes. In particular, relationships can be easily queried as easily as nodes, allowing for data access solely based on links rather than data.

graphs - gremlin:
As a database designed for graph models, gremlin is poor with relational data and highly efficient with graph data. In comparison with neo4j, the gremlin model is similar in the way it models edges ('relationships') and vertices ('nodes') with high flexibility. The two models are equally expressive.
Compared to XML based models, it is more expressive when it comes to modeling graph data as many to many relationships can more easily be modeled by simply connecting nodes with edges, while any XML based model will be constrained by the tree hierarchy.

JSON - N1QL (Couchbase's QL):
Very similar to XML although the latter is more verbose. XML does not include arrays while JSON does.
JSON is not ideal for relational models where referential integrity is needed although Couchbase does provide the notion of primary key (unlike in XML). This model is ideal for large amounts of semi-structured data.

2 Query Languages
2.1 Is the QL procedural or declarative? Procedural means that it explicitly states the operations (and their order) required to find the data. Declarative QLs only specify what conditions the data need to satisfy to be returned by the query, without any hint on how to find this data.

Relational Model - SQL:
The QL is declarative - with SQL one can specify the outcome of the query, and the database query optimizer decides what is the best set of procedures to satisfy the query.

OO - OQL:
Functional: the desired computation is specified by composing functions — so it's a type of declarative language.

XML - XQuery:
Functional: the desired computation is specified by composing functions — so it's a type of declarative language.

neo4j graphs - Cypher:
Declarative query language.

graphs - gremlin:
Declarative language

JSON - N1QL (Couchbase's QL):
Declarative language.

2.2 What kind of navigation is supported?
2.2.1 How can one access the properties of an entity? Think of the analogous record projection in standard programming languages. Call this access a navigation step. In particular, it is a projection step.

Relational Model - SQL:
The navigation/projection step takes the form of a series of predicates specified as conditions onto relations in the SQL language. Conditions can characterize the whole relation, a particular subset, or can span multiple relations by means of join conditions.

OO - OQL:
One can access entities and their properties through path expressions.

XML - XQuery:
The FLOWR expression allows to navigate and access nodes and node properties. The navigation occurs by traversing down a particular tree path by entering it at a specified node and selecting output nodes and related properties by specifying a predicate.

neo4j graphs - Cypher:
As entities can be both nodes or relationships, we can directly query either. This means we can access data by starting with relationships or with nodes, thus allowing easier predicate constructs.

graphs - gremlin:
Navigation is performed by graph traversal. One starts at a specified node and 'walks' along an edge to reach other nodes. Once the target node is reached one can access its properties.

JSON - N1QL (Couchbase's QL):
One can access the properties of an entity by accessing the entity itself via the QL.

2.2.2 If relationships between entities are supported, can one access an entity from another entity by following a relationship between them? Is the crossing of a relationship link between two entities possible? Classify this crossing also as navigation step, in particular a traversal step.

Relational Model - SQL:
Although relationships can be defined with referential integrity, more generally in SQL entities can be accessed from others by means of joins which do not require the enforcement of a specific

relationship. The join condition can be submitted upon any combination of columns and the comparison will be value based (by means of atomic values) - so as long as at least some tuples match based on the specified join condition, the access can be made.
In SQL it is possible to access different relations from within the same access request (query) by specifying multiple join conditions.

OO - OQL:
Yes this is possible through the use of a path expression. The path expression starts at a persistent object name.

XML - XQuery:
The FLOWR expression allows to navigate and access nodes and node properties. The navigation occurs by traversing down a particular tree path by entering it at a specified node and selecting output nodes and related properties by specifying a predicate.

neo4j graphs - Cypher:
Yes - and the traversal step is possible.

graphs - gremlin:
Yes - and the traversal step is possible.

JSON - N1QL (Couchbase's QL):
Yes - although relationships are not mere objects as in the case of graph database models but just nested parent-child hierarchy paths.

2.2.3 Are relationships directed or undirected? That is, can relationships be crossed in either direction, or are they asymmetric?

Relational Model - SQL:
Relationships (i.e. joins) are undirected. Although outer (left or right joins) can be specified as predicates, the crossing of tables is symmetric.

OO - OQL:
Relationships are directed - one starts at a persistent object and navigates in a specific direction to another object.

XML - XQuery:
The navigation is directed. One can navigate both 'up' or 'down' the tree path by starting at a specific node. From a syntactic perspective navigating to a child is done by specifying child nodes and attributes, while navigating to a parent might require using built in functions.

neo4j graphs - Cypher:
Relationships are directed and can be crossed in either direction.

graphs - gremlin:

Edges are directed and can be crossed in either direction by means of a step. Nearly every step in GraphTraversal either extends MapStep, FlatMapStep, FilterStep, SideEffectStep, or BranchStep.

JSON - N1QL (Couchbase's QL):
Relationships are directed because of the tree structure.

2.2.4 What corresponds to the notion of entry point into the data? How do we specify the entities from which we wish to start navigating the data?

Relational Model - SQL:
The entry point into the data is the relation(s) specified in the SQL 'FROM' clause. Additional syntax can be implemented in the query to specify a subset of the entities contained in the relations included in the 'FROM' clause, such as the 'Select' clause which permits the specification of predicate variables.

OO - OQL:
The entry point is the extent name i.e. the class.

XML - XQuery:
The entry point to the data is the node. This can be retrieved by either specifying it directly via identifier or label, or by implementing a predicate condition as part of a FLWR expression.

neo4j graphs - Cypher:
Entry points are either nodes or relationships. To start from a specific point we simply place the intended point in the query syntax and draw directed relationship paths outward to the intended target.

graphs - gremlin:
The entry point to the data is the graph traversal. Once the graph traversal is instantiated, Edge 'walks' always start with nodes (preceded by the specification of the graph as the very first step in the expression).

JSON - N1QL (Couchbase's QL):
The entry point is the document which is specified in the path expression as part of the FROM clause.

2.2.5 Is there a notion of path expression? A path expression describes a (possibly singleton) sequence of navigation steps. A path expression evaluated at entity e returns the collection of entities reachable from e via the sequence of steps described by the expression.

Relational Model - SQL:
Path expressions are typical of graph database models as they enable the access of a particular entity by specifying a hierarchical path from the data entry point to the specified destination.

From the logical perspective the relational model uses joins as equivalent to path expressions, but relationships are not managed as entities so technically there is no notion of path expression.
N/A

OO - OQL:
Yes

XML - XQuery:
The FLWR expression is a path expression.

neo4j graphs - Cypher:
Cypher has paths - a path is a walk through a property graph and consists of a sequence of alternating nodes and relationships.

graphs - gremlin:
The graph traversal is the primary means by which walks from one node to another are performed. Similar to a path expression, the traverser is created by executing an expression composed by the source of the traversal on the left of the expression (e.g. vertex 1), the steps in the middle of the traversal (e.g. out('knows') and values('name')), and the results "traversal.next()'d" out of the right of the traversal

JSON - N1QL (Couchbase's QL):
Yes - path expressions are used to navigate the tree hierarchy of the model. Path expressions are inserted in the SELECT clause to prefix the needed outputs with the data structure where the outputs are stored.

2.2.6 Is it legal for the result of a path expression starting at a source entity to be a collection of target entities?

Relational Model - SQL:
N/A

OO - OQL:
Yes - the path expression can specify a directed link to any object satisfying a set of predicate conditions thus returning more than one object as query result. The OQL query returns a collection.

XML - XQuery:
Yes - the FLWR expression can retrieve a collection of nodes or node attributes as specified in the expression's predicate.

neo4j graphs - Cypher:
Yes - the traversal of a path can return more than one node. Paths always start and end at a node.

graphs - gremlin:

Yes - the traversal can result in a collection of nodes.

JSON - N1QL (Couchbase's QL):
Yes - multiple target entities can be returned by a query.

2.2.7 Is it legal for the result of a path expression to be the empty collection?

Relational Model - SQL:
N/A

OO - OQL:
No - if the object is empty an exception is raised.

XML - XQuery:
Yes - empty node sets can be returned legally.

neo4j graphs - Cypher:
Yes - technically this is defined as the empty path which contains only the starting node.

graphs - gremlin:
Yes - the returned traverser can be empty.

JSON - N1QL (Couchbase's QL):
Yes - the returned output is empty.

2.2.8 If the results of path expressions are sets of entities, how are duplicates removed?

Relational Model - SQL:
N/A

OO - OQL:
The results of a path expression cannot be sets of complete duplicate entities but will always be bags because each instantiated class type is unique at least with respect of one class attribute. However if the specifications of query predicates results in duplicate query outputs it is possible to use query syntax to remove them from the output (for example 'group by' operators).

XML - XQuery:
Path expressions automatically remove duplicates (the set is turned into a sequence). This is down as part of the 'union' final step in the path expression.

neo4j graphs - Cypher:

Entities are assigned a unique identifier so true duplicates in the sense of a SQL relation where uniqueness is not enforced does not apply. However, a path can return duplicate values in which case the MERGE or other operators can be used to remove them from the query output.

graphs - gremlin:
Duplicate outcomes are allowed but the 'toSet()' step can be used to remove them.

JSON - N1QL (Couchbase's QL):
The DISTINCT or GROUP BY clause syntax are the easiest ways to omit duplicate outputs.

2.2.9 How are duplicates defined? by value- or by identity-eqality?

Relational Model - SQL:
N/A

OO - OQL:
Duplicates can only exist in terms of atomic values as every instantiated type is unique with respect to a key or type identifier.

XML - XQuery:
Duplicates are defined by value.

neo4j graphs - Cypher:
As mentioned above duplicates can only exist in the sense of entities with identical property values but will always be assigned a unique ID so true duplicates are not allowed. Therefore, duplicates are defined by value.

graphs - gremlin:
Duplicates are defined by value as each entity is truly unique because it is defined by a unique key.

JSON - N1QL (Couchbase's QL):
Duplicates are defined by value if the primary key is set via Couchbase. Otherwise, duplicate entities can exist within the same document - identical in all ways as long as data structures are identical.

2.2.10 Are path expressions structure-preserving or flattening? Suppose that the result of evaluating path expression p1 starting from entity e is a collection C of entities. Suppose that for each of the entities in C, the result of evaluating path expression p2 is in turn a (possibly non-empty) collection of entities. Then what does the path p obtained by concatenating p1 with p2 return when evaluated at e? A collection of collections? Or is this result a single collection, obtained by merging the nested collections? In the former case, we say that path expressions are structure-preserving, in the latter we say that they are flattening.

Relational Model - SQL:
N/A

OO - OQL:
Path expressions are flattening - the result of multiple path expressions is in effect a merge of multiple path expression outputs.

XML - XQuery:
Path expressions are flattening - the result of multiple path expressions is in effect a merge of multiple path expression outputs.

neo4j graphs - Cypher:
Paths are flattening

graphs - gremlin:
Traversers are flattened

JSON - N1QL (Couchbase's QL):
Path expressions are structure preserving - they will return nested objects if use to query any.

2.2.11 Can path expressions traverse collections? If a navigation step s starting from entity e returns a collection, are path expressions starting with s and continuing with at least one more step legal at e?

Relational Model - SQL:
N/A

OO - OQL:
Yes - as long as the needed relationships to allow the navigation steps exist.

XML - XQuery:
Yes - as long as the needed relationships to allow the navigation steps exist.

neo4j graphs - Cypher:
Yes - a path can start at a node and go as far as possible as long as relationships allowing to do so exist.

graphs - gremlin:
Yes

JSON - N1QL (Couchbase's QL):
Yes - path expressions will simply access any nested object or property as long as they follow the desired path.

2.2.12 Can path expressions specify traversal along unboundedly many navigation steps? As opposed to bounded by the size of the expression?

Relational Model - SQL:
N/A

OO - OQL:
The number of navigation steps is bounded by the size of the expression.

XML - XQuery:
The number of navigation steps is bounded by the size of the expression.

neo4j graphs - Cypher:
Unbounded traversals are allowed

graphs - gremlin:
Unbounded traversals are allowed

JSON - N1QL (Couchbase's QL):
Unbounded traversals are allowed.

2.3 Does the language have variables bound to the results of path expressions?

Relational Model - SQL:
N/A

OO - OQL:
It does - iterator variables are used to reference a collection output by an OQL query and to then range over each object in the collection e.g. 'd in departments'

XML - XQuery:
Yes - variables are used to specify outcomes obtained within the query so they can be in turn used to accomplish additional query steps.

neo4j graphs - Cypher:
Yes - variables can be used as intermediate holders to contain path step results and apply additional predicate conditions to construct complex path queries.

graphs - gremlin:
Yes - variables can be used to store the nodes output by graph traversals

JSON - N1QL (Couchbase's QL):
The JavaScript constructor is in fact a variable used to instantiate a type. The constructor can be used to instantiate objects of any data structure.

2.3.1 Can path expressions start from previously defined variables, or do they have to start from entry points only?

Relational Model - SQL:
N/A

OO - OQL:
They can also start from previously defined variables

XML - XQuery:
They can also start from predefined variables - this is often done to navigate through a tree path obtained as a result of another expression.

neo4j graphs - Cypher:
Yes - they can start from predefined variables.

graphs - gremlin:
The traversal has to start by instantiating a traversal and specifying the vertex. However, one can pass a variable storing another traversal's outcome into the vertex and directly access the data from there.

JSON - N1QL (Couchbase's QL):
The entry point is always the document. The data structure object can then be specified within the SELECT clause as a variable, but it is not the entry point.

2.3.2 Can variables be compared to each other and to constants? For equality comparisons, what kind of equality is supported? Value-based or identity-based?

Relational Model - SQL:
Variables i.e. query predicates are compared by value. Queries can include syntax to pass constant values for value based comparisons.

OO - OQL:
Variables can be used for comparison purposes within 'WHERE' clauses. Equality can be evaluated with respect to value by means of the collection interface or additional collection type specific interfaces.

XML - XQuery:

Yes - variables can be used to traverse paths by setting equality conditions (predicates). Both value and identity based equality are allowed, it depends on the predicate logic specified within the expression.

neo4j graphs - Cypher:
Yes - variables can be leveraged while traversing paths to hold outcomes from different path traversals so they can be used as intermediary steps within movement through other paths. Variables can be assigned to both nodes and relationships. The MATCH method can then be used to compare results held in variables.

graphs - gremlin:
Variables can be used to store data and for value comparison and computation purposes. Variables can also be used to store the outcome of traversals - in this case the comparison is made by identifier (entity key).

JSON - N1QL (Couchbase's QL):
Yes - comparing variables is to compare objects.

2.3.3 Can one give a pattern-match/construct semantics to queries? Can one conceptually think of queries as performing a two-phase computation as follows? The query is specified by a pattern, possibly containing variables, and a return expression. In the first phase, the pattern is matched in all legal ways against the data. Each match yields a tuple of values providing the binding for the tuple of pattern variables. Call match table the collection of the tuples yielded by the match phase. In the second phase the return expression is executed for each tuple of the match table. If the return expression depends on the variables, the match tuple provides the corresponding variable binding as input to the return expression.

Relational Model - SQL:
Conceptually SQL SELECT queries can be thought of as perfuming the above described two phase query as FROM clauses are first evaluated to return a set of relations containing entities that are candidate for a second phase (the 'Return' phase) where a set of predicate conditions is applied to the candidates to filter entities before the phase is completed.

OO - OQL:
Similarly to SQL, OQL constructs queries by composing outputs with an entry point access step and a predicate step. Multiple queries can be nested within others, adding more compositional options to query development possibilities.

XML - XQuery:
Yes - FLWOR expressions technically navigate through node paths by first returning all nodes meeting a specific navigation predicate and then selecting and/or performing additional steps on intermediate results. Additionally, FLWOR expressions can be nested within others in order to perform more complex navigation steps.

neo4j graphs - Cypher:

Yes - one can think of cypher queries as compositional navigation steps finalized by a return step where the path outcome is output.

graphs - gremlin:
Yes - traversals perform the walk by executing steps after starting with a specified traversal and node. Then, the 'next()' step returns the outcome of the traversal.

JSON - N1QL (Couchbase's QL):
Syntactically similar to SQL, Couchbase perform a data access step within the FROM clause, and a return step within the SELECT clause (or execute actions as specified if an alternative to the SELECT clause is implemented).

2.4 Is the language compositional? Does the result of queries belong to the same data model as the query inputs?

Relational Model - SQL:
The language is not fully compositional as composition rules are not orthogonal to the type system. The results of queries will always belong to the same data model.

OO - OQL:
Yes, OQL is a functional language and all operators can be composed freely as long as the type system is preserved. The data type of a query result can be any type defined in the ODMG model (so yes they can belong to the same data model as the query inputs).

XML - XQuery:
Yes, XQuery is a functional language and all operators can be composed freely as long as the type system is preserved. The data type of a query result can be any type defined in the ODMG model (so yes they can belong to the same data model as the query inputs).

neo4j graphs - Cypher:
Yes - graph query results can be saved into variables and reused by other path expressions.

graphs - gremlin:
Yes - traverses can be stored into variables and reused by other traversers. Steps can be used to compose and decompose traversers.

JSON - N1QL (Couchbase's QL):
The language is compositional - one can compose query statements from within others.

2.5 If the language is compositional, and the model is complex-valued, can the query construct complex-values?
What constructors are there for building the output complex values from input complex values?

Relational Model - SQL:
N/A

OO - OQL:
The language is compositional and the model is complex valued; queries can construct complex values by composing queries with multiple nested select statements involving complex value structures and returning outputs that are compositional types. Explicit output structures can be specified directly as part of the query.

XML - XQuery:
The language is compositional and the model is complex valued; queries can construct complex values by composing queries with multiple nested FLWOR expressions selecting values from many nodes and performing additional query steps on them.

neo4j graphs - Cypher:
N/A - language not complex valued.

graphs - gremlin:
N/A language is not complex valued.

JSON - N1QL (Couchbase's QL):
The query DDL does allow to construct complex values.

2.5.1 What are the semantics of executing a call to a constructor taking a variable as argument? Value- or reference-based? Suppose a constructor is called with a variable x as argument, with the intended meaning that a new entity is built whose contents is given by the bindings of x. What exactly does "given by" mean here? Is x output by value, i.e. does the constructor use a copy of the value x binds to (deep copy if the value is complex)? Or does the constructor output x by reference, i.e. outputs the identity of x's binding (of course this only makes sense if the data model has a notion of identity).

Relational Model - SQL:
Not being an Object Oriented Programming Language, SQL does not include variables as part of its relation creation features.

OO - OQL:
The constructor is semantically added explicitly to the query, e.g. 'struct' in the following example:
'select struct (employee: e.name, ….) from e in Globe.employees'
The constructor will output of x by reference.

XML - XQuery:
An entity is declared in the document type and then referenced.

neo4j graphs - Cypher:

The CREATE statement syntax is the closest function to a pure OOP constructor in cypher. One can create additional entities directly by writing them in JSON syntax and then pass the contents as a parameter to the CREATE statement. More complex DDL statements include the SET method which also enables updates.
Here, the constructor uses a copy of the value x (the parameter) binds to.

graphs - gremlin:
Call by reference.

JSON - N1QL (Couchbase's QL):
Constructor calls are reference based and use the standard 'var' JavaScript constructor.

2.6 Can constructor calls take as argument collections of entities returned by queries? We call these queries "nested" within the "outer query" that performs the constructor call.

Relational Model - SQL:
N/A

OO - OQL:
Constructor calls can take embedded structures by combining struct operators and nesting them if needed.

XML - XQuery:
Yes

neo4j graphs - Cypher:
Yes - one can pass more complex arguments by for example passing procedures to the constructor.

graphs - gremlin:
Yes - a constructor can be passed a variable storing a traverser as argument.

JSON - N1QL (Couchbase's QL):
No - constructors can only instantiate data structures defined ad hoc.

2.6.1 Is the nesting depth of queries bounded?

Relational Model - SQL:
N/A

OO - OQL:
It is not - one can apply as many nested struct or select operators as wanted.

XML - XQuery:

It is not - one can apply as many nested FLWOR expressions as wanted.

neo4j graphs - Cypher:
No - one can pass as many variables as wanted, each in turn holding additional variables representing paths.

graphs - gremlin:
No - unbounded function nesting is part of Gremlin's core functionality

JSON - N1QL (Couchbase's QL):
No

3 The Survey Answer all the above questions for the following (model,language) pairs studied during this course:
model language
relational SQL
OO OQL
XML XPath
XML XQuery
neo4j graphs - Cypher
graphs - gremlin
JSON - N1QL (Couchbase's QL)

Refer to the questions via the section numbers whose titles they appear in. Feel free to answer N/A or 'it depends' as needed. In the latter case, give a very brief justification.