CS5200
Database Management
Object-Relational Mapping

Ken Baclawski
Spring 2017

Outline

- Model Mappings
- Schema Customizations
  - Mapping File
  - Annotations
- Object Synchronization
- Query Language
- Assignment #10
- Review for Final Exam

Model Mappings

Relational versus Object-Oriented

- Programming languages primarily use objects
- Relational databases use rows
- Mapping between them
  - Data point of view (JDBC)
  - Program point of view (JPA)
- Mapping Levels
  - Classes and Tables
  - Objects and Rows
  - SQL Queries and Object-Oriented Queries

Object-Relational Mapping Software

- Automating the mappings between object-oriented and relational
- Separate mapping file
  - More flexible
  - Separates concerns
- Annotations
  - Only one mapping
  - Closer integration with the source code

Schema Customizations with Annotations

Annotations

- Targets
  - Package
  - Type
  - Field
  - Method
  - Constructor
  - Local Variable
  - Parameter
  - Annotation
- Retention
  - Source only
  - Class file
  - Runtime
- Inherited

JPA Annotations

- javax.persistence
  - ~90 annotations
  - ~20 enumerations
  - 16 interfaces
  - 1 class
- Few annotations are necessary if the default mapping is used
- Most annotations are for customizing the mapping
- Plurals
  - Same annotation type more than once
  - Multiple annotations of the same type are now supported
  - Example: JoinColumns

Navigation

- An association can be unidirectional or bidirectional
- One side of a bidirectional association is the owner
- UML specifies navigability by using an arrow on the association line
- If an association has no arrow, then the association is bidirectional
- UML does not have a notation for ownership

Access

- A field can be accessed either directly or via getters and setters
- The getter and setter for a field form a *property*
- Access should be specified as being either `FIELD` or `PROPERTY`

Fetching

- Objects are fetched into memory as a result of a query
- Should the attributes and associations also be fetched into memory?
  - Those would also fetch other objects, and so on
  - This could easily exceed the amount of memory
  - It also takes a lot of time
- Each attribute and association can be specified to be
  - `EAGER` which automatically fetches the value(s)
  - `LAZY` which only fetches the value(s) when requested

Student Class

```
@Entity
@Access(FIELD)
public class Student {
    @Basic
    public String name;

    @Basic
    public String email;

    @Basic @Temporal(DATE)
    public Date matriculation;
```

Student Class

```
@ManyToMany(cascade=ALL)
@JoinTable(
  name="Enrollment",
  joinColumns=@JoinColumn(name="student"),
  inverseJoinColumns=@JoinColumn(name="course")
)
public Set<Course> enrollment = new HashSet<Course>();

public Student() {}
```

The name of the join table

The foreign key columns in the join table

The Student class is the owner of the enrollment association

Student Class

```
    public Student(String name, String email) {
        this.name = name;
        this.email = email;
        this.matriculation = new Date();
    }

    public String toString() {
        return "Student " + name + " " + email +
                " year " + matriculation;
    }
}
```

Course Class

```
@Entity
@Access(FIELD)
public class Course {
    @Id @GeneratedValue
    public int code;

    @Basic
    public String name;

    @Basic
    public String description;

    @ManyToMany(mappedBy="enrollment", cascade=ALL)
    public Set<Student> roster = new HashSet<Student>();
```

Course Class

```
    public Course() {}

    public Course(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public String toString() {
        return "Course " + code + " " + name +
                " " + description;
    }
}
```

Class Annotations

@Entity The class is to be persistent

If you need to change the default mapping use

@Table Specify the primary table of an entity and other properties such as uniqueness constraints

The @Table annotation (and some others) use the following

@UniqueConstraint Unique constraint on set of columns
@Index An index on a set of columns
@Access Whether to use field or property access by default
@Cacheable Whether the entity should be cached
@SecondaryTable(s) Partition the class into multiple tables

Field/Property Annotations

These apply to fields that have a built-in type

@Id Primary key column
@Basic Basic column type information
@Column More complete column type information
@Transient The column is not to be made persistent

@Lob Used for BLOB, CLOB and NCLOB
@Temporal The type used for SQL time
@GeneratedValue Automatically generated column

Multi-Valued Attribute Annotations

These apply to fields that have a built-in type

@ElementCollection The attribute is a multi-valued attribute

If you need to change the default mapping use

@CollectionTable The multi-valued attribute table name
@JoinColumn(s) Specify the column name within
the collection table

As an alternative to the above, one can explicitly define a class that is to map to the collection table, then use the following

@MapsId

Association Annotations

@ManyToMany
@ManyToOne
@OneToMany
@OneToOne

These specify the main cardinalities of a field whose type is another class

@JoinTable The table for many-to-many association

@JoinColumn(s) Specify the column name(s) within the join table

@OrderBy Ordering of a collection upon retrieval
@OrderColumn Ordering of a collection persistently

@ForeignKey Explicit control over enforcement of foreign key constraints

Conversion Annotations

These are used for converting attribute and association values between the values in memory and the values in the database fields

@Convert(s) Specify how to convert
an attribute or association
@Converter A class used for conversion

Inheritance Annotations

@Inheritance The inheritance strategy

@DiscriminatorColumn This is for the single table strategy
@DiscriminatorValue The discriminator value of this class
for the single table strategy

@PrimaryKeyJoinColumn(s)
Change the name(s) of the join column(s) for
the joined strategy and in some other cases

More Inheritance Annotations

@MappedSuperclass
Mapping information in the superclass (that is not persistent) for subclasses that are persistent

@AssociationOverride(s) These override mapping
@AttributeOverride(s) information specified in the mapped superclass

Composite Column Annotations

A class-valued attribute can be mapped as a set of columns in the table for the containing class rather than as a foreign key to another table. This is called an embedded class. It is primarily used for defining multi-valued primary keys.

@Embeddable A class to be embedded in another class
@Embedded An attribute that is to be mapped by embedding rather than with a foreign key
@EmbeddedId Same as @Embedded but primary key

@IdClass A simpler way to specify a multi-column primary key

Observer Annotations

One can specify methods that are to be invoked when objects are made persistent, loaded, removed and updated. These are called listeners or callbacks.

@EntityListeners The classes that have listeners for this class

@PrePersist Invoke before the event
@PreRemove
@PreUpdate

@PostLoad Invoke after the event
@PostPersist
@PostRemove
@PostUpdate

Query Annotations

Queries can be specified with annotations. This is more efficient than specifying them within the Java code.

@NamedQuery(ies) Specify JPQL query or queries
@NamedNativeQuery(ies) Specify SQL query or queries
@NamedStoredProcedureQuery(ies)
Specify a vendor-specific stored procedure

These use the following annotations

@QueryHint Vendor-specific query hint
@StoredProcedureParameter
Parameter for a stored procedure

Query Annotations

One can specify how the results of a query are mapped

@SqlResultSetMapping Specify a named query mapping

The mapping annotation uses the following

@ColumnResult Column alias
@ConstructorResult Construct an object from the results
@EntityResult Entities in a result of a query
@FieldResult Map field results

Eager/Lazy Loading Annotations

The fetch attributes of various annotations control how that attribute is to be loaded, but it does not specify anything else. A more sophisticated way to specify eager loading is to use entity graphs

@NamedEntityGraph(s) Specify eager loading graph(s)

This annotation uses the following

@NamedAttributeNode Member of an entity graph
@NamedSubgraph Subgraph of an entity graph

Persistence Unit Annotations

Entity manager factories are like JDBC connections. Entity managers perform the actual work. The properties of the connection and manger can be specified with annotations.

@PersistenceUnit(s) Database unit(s) for a manager factory
@PersistenceContext(s) Context(s) for a manger

The context annotation uses

@PersistenceProperty Vendor-specific property

Map Mapping Annotations

These are used for mapping an attribute of type `java.util.Map`

@MapKey
@MapKeyClass
@MapKeyColumn
@MapKeyEnumerated
@MapKeyJoinColumn(s)
@MapKeyTemporal

Miscellaneous Annotations

One can specify automatically generated primary keys in several ways, if one is not using the default generator

@SequenceGenerator
@TableGenerator

A column that is to be used for non-blocking (optimistic) concurrency control may be specified using

@Version

Schema Customizations with a Mapping File

Hibernate Customization File

- Used prior to the development of JPA annotations
- No standard
- Poorly documented
  - Only some informal tutorials
- More verbose than annotations
- Fewer features than annotations
- No longer in active development by Hibernate

Hibernate Mapping File

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="nu" default-access="field">
  <class name="Student">
    <id name="studentId">
      <generator class="native"/>
    </id>
    <property name="name"/>
    <property name="email"/>
    <property name="matriculation" type="date"/>
    <set name="enrollment" table="Enrollment"
         cascade="all">
      <key column="student"/>
      <many-to-many column="course" class="Course"/>
    </set>
  </class>
```

Hibernate Mapping File

```
  <class name="Course">
    <id name="code">
      <generator class="native"/>
    </id>
    <property name="name"/>
    <property name="description"/>
    <set name="roster" table="Enrollment" cascade="all"
         inverse="true">
      <key column="course"/>
      <many-to-many column="student" class="Student"/>
    </set>
  </class>
</hibernate-mapping>
```

Mapping File vs Annotations

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="nu" default-access="field"> @Access(FIELD)
  <class name="Student"> @Entity
    <id name="studentId"> @Id
      <generator class="native"/> @GeneratedValue
    </id>
    <property name="name"/> @Basic
    <property name="email"/> @Basic
    <property name="matriculation" type="date"/> @Basic @Temporal(DATE)
    <set name="enrollment" table="Enrollment" @ManyToMany(cascade=ALL)
         cascade="all"> @JoinTable(name=”Enrollment”)
      <key column="student"/> @JoinColumn(name=”student”)
      <many-to-many column="course" class="Course"/>
    </set>           @JoinColumn(name=”course”)
  </class>
```

Mapping File vs Annotations

```
  <class name="Course"> @Entity
    <id name="code"> @Id
      <generator class="native"/> @GeneratedValue
    </id>
    <property name="name"/> @Basic
    <property name="description"/> @Basic
    <set name="roster" table="Enrollment" cascade="all"
         inverse="true"> @ManyToMany(mappedBy=”enrollment”,
      <key column="course"/>          cascade=ALL)
      <many-to-many column="student" class="Student"/>
    </set>
  </class>
</hibernate-mapping>
```

Object Synchronization

Terminology

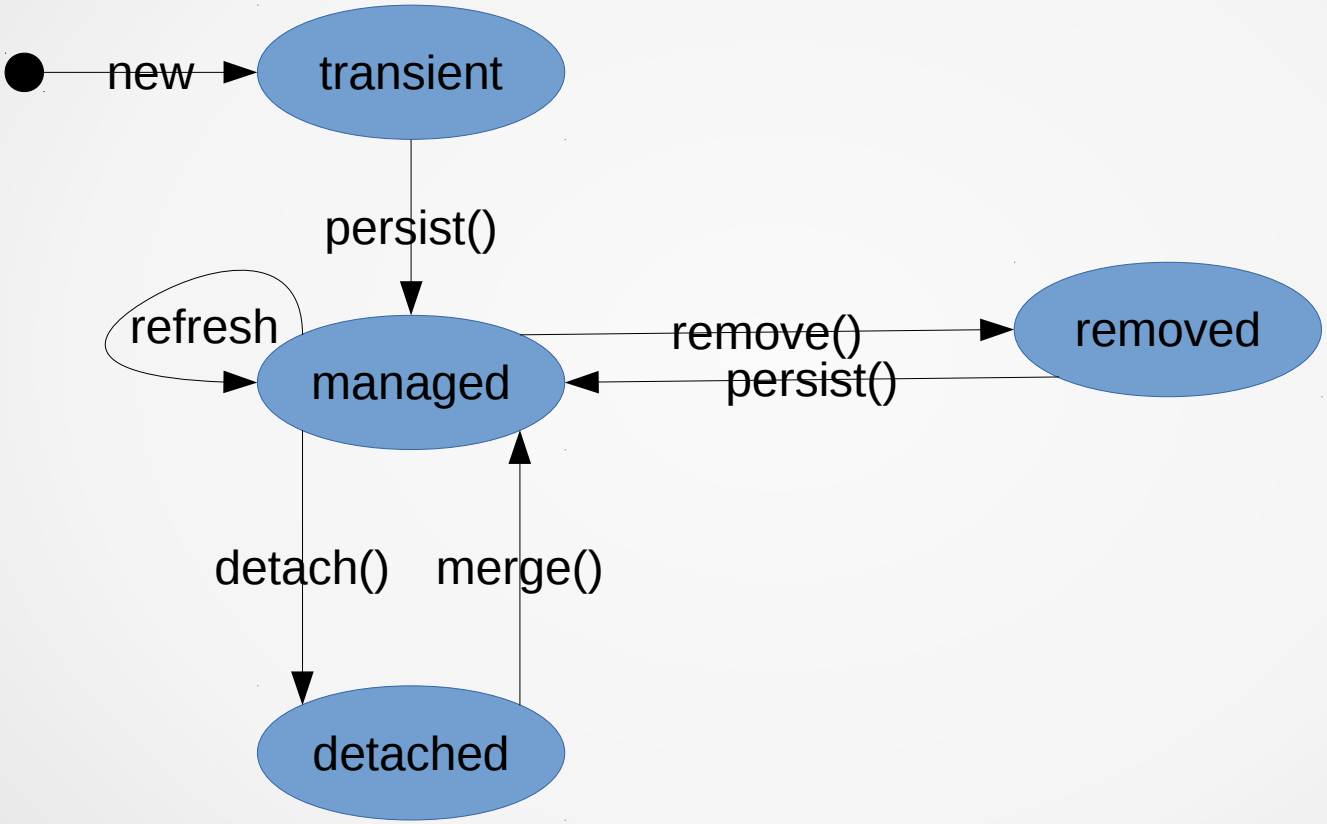
- Objects are both in memory and in the database
  - An object in the database is *persistent*
  - An object in memory can be in one of four states:

    new, managed, removed or *detached*
- A new object has been created but is not persistent and not known to the entity manager
- A managed object is persistent and known to an entity manager
  - Updating a managed object also updates the persistent object

Terminology

- A removed object will be deleted from the database
- A detached object is still persistent and but no longer managed by an entity manager
  - Updating a detached object does not update the persistent object
- Refreshing a managed object causes its state to be updated to be consistent with the persistent object

Object Lifecycle

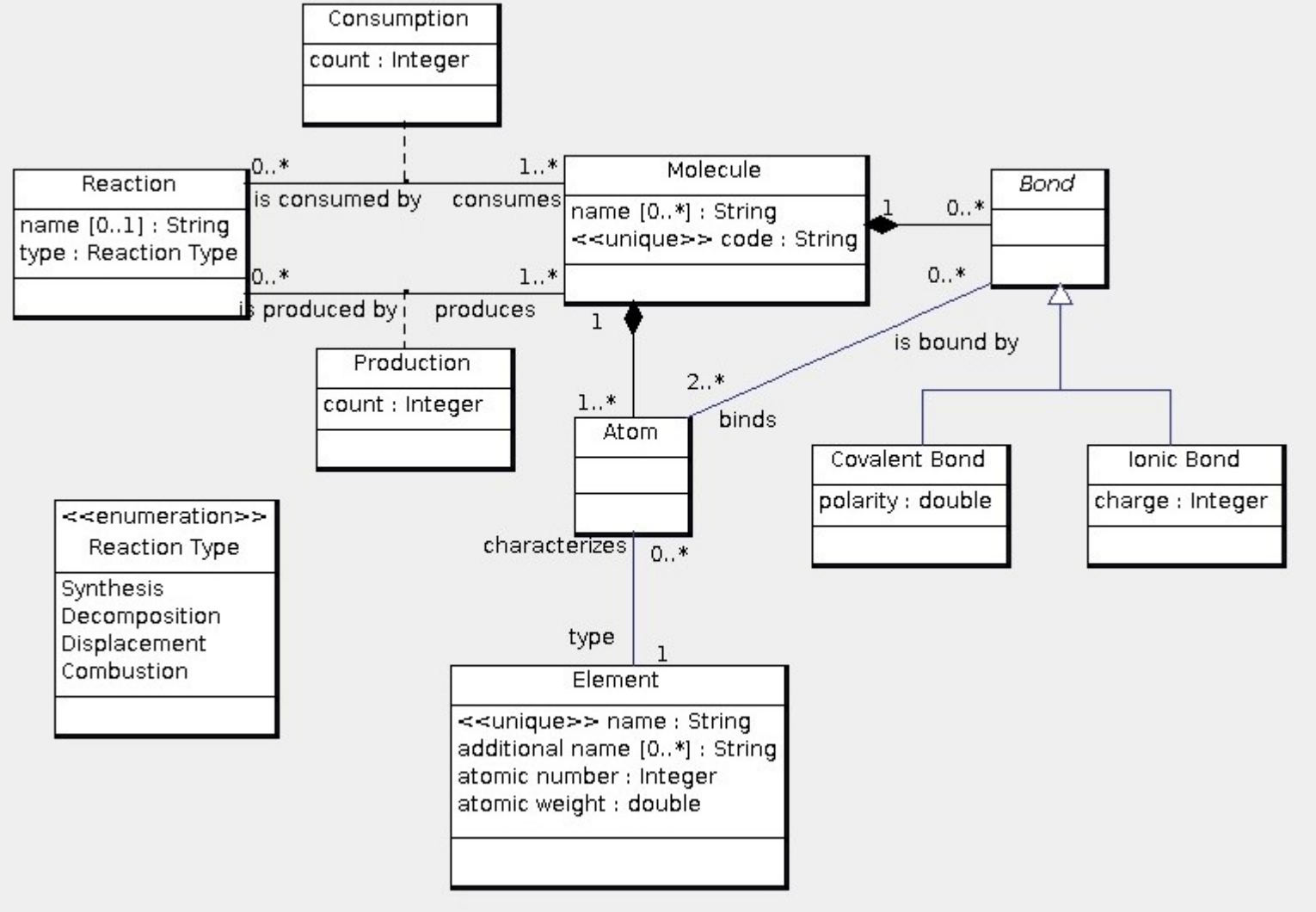


Object Query Language

JPQL Query Language

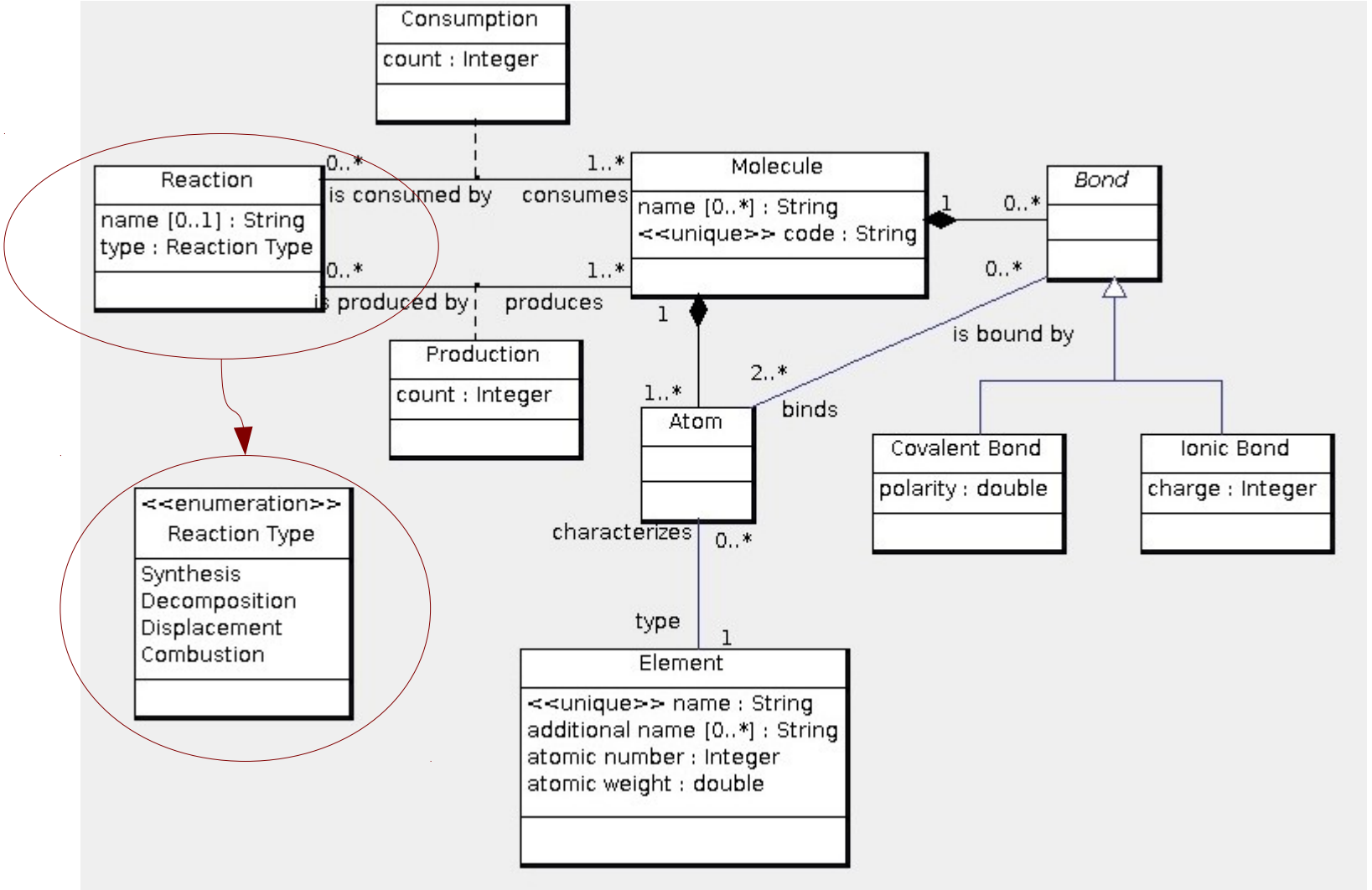
- JPQL uses an SQL-like syntax to allow direct use of objects and relationships
- Queries have either positional or named parameters
- Unlike SQL, JPQL queries can arbitrarily long path expressions
- The examples are taken from Lectures 03 and 04.

Example queries will use the chemistry database



Query 1

- What are the ids, names and types of all reactions?
- In JPQL one can return the objects rather than the attributes of the objects



Query 1

What are all reactions?

```
create table Reaction (
  id int primary key,
  name varchar(200),
  type enum ('Synthesis',
    'Decomposition',
    'Displacement',
    'Combustion') not null
);
```

Answer:

```
select r
  from Reaction r
```

Query 2

What are the combustion reactions?

```
create table Reaction (
  id int primary key,
  name varchar(200),
  type enum ('Synthesis',
    'Decomposition',
    'Displacement',
    'Combustion') not null
);

Answer:

select r
  from Reaction r
 where r.type='Combustion'
```

Query 3

What are the combustion reactions that have a name?

```
create table Reaction (
  id int primary key,
  name varchar(200),
  type enum ('Synthesis',
    'Decomposition',
    'Displacement',
    'Combustion') not null
);

select r
  from Reaction r
 where r.type='Combustion'
   and r.name is not null
```

Query 4

What are the reactions that have a name that begins with 'Aldol'?

```
create table Reaction (
  id int primary key,
  name varchar(200),
  type enum ('Synthesis',
    'Decomposition',
    'Displacement',
    'Combustion') not null
);

Answer:

select r
  from Reaction r
 where r.name like 'Aldol%'
```

Query 5

- List all synthesis reactions
- In SQL one cannot return the actual reaction objects
- In JPQL one can return objects
- The SQL and JPQL queries are:

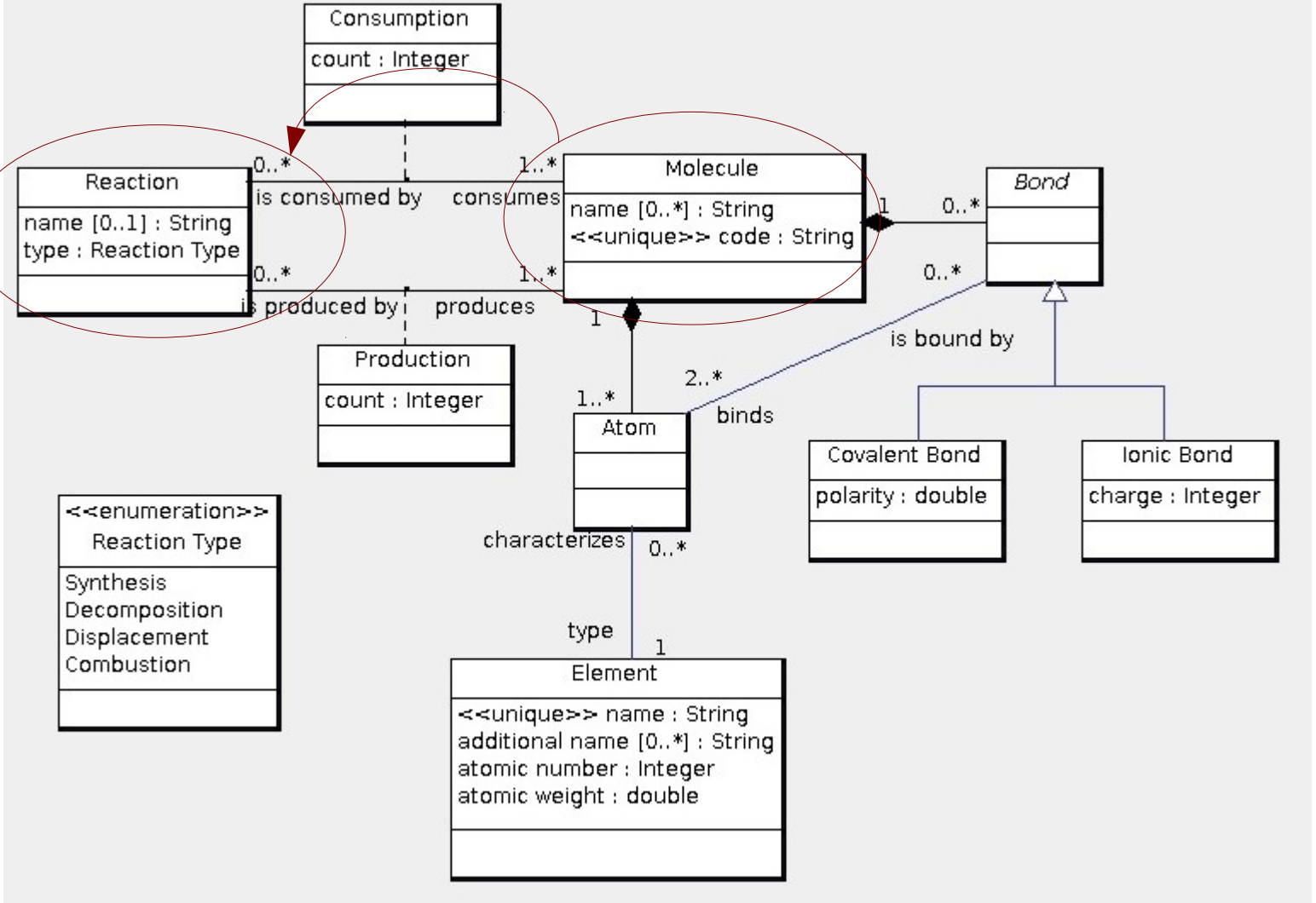
```
select id
  from Reaction
 where type='Synthesis'
```

```
select r
  from Reaction r
 where r.type='Synthesis'
```

Query 6

- List all molecules consumed by a reaction named WFO
- JPQL joins may navigate using the fields/properties rather than only by columns
- JPQL allows using the dot notation any number of times
  - One can continue as long as the attribute is single-valued
  - If an attribute is multi-valued, then use a join

Navigation for Query 6



Query 6

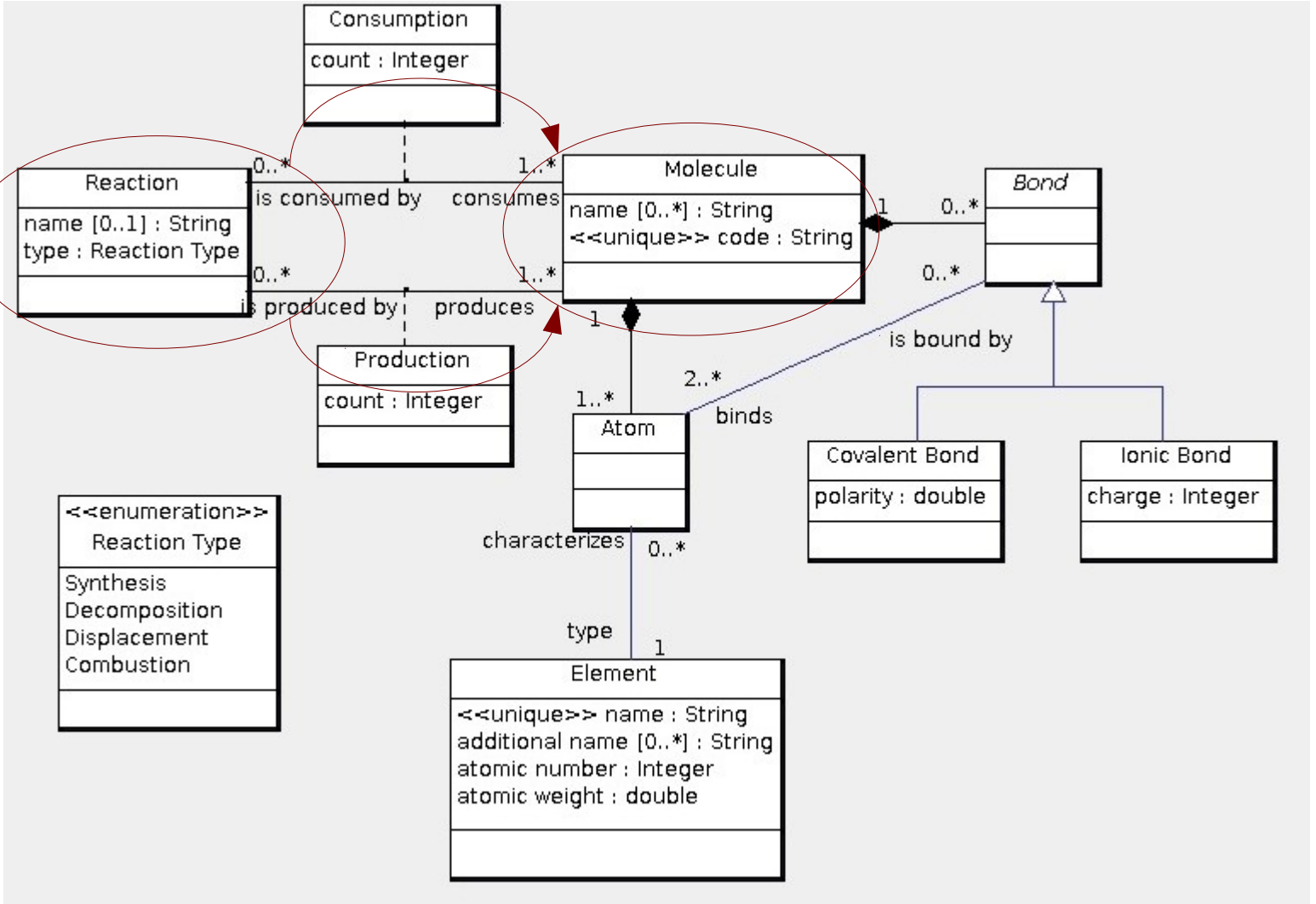
- The query navigates to the Consumption class
  - This is one-to-many
- Then it navigates to the Reaction class
  - This is many-to-one
- Finally, navigate to the name attribute
- The query is:

```
select m
  from Molecule m join m.isConsumedBy c
 where c.reaction.name='WFO'
```

Query 7

- List the molecules by code that are consumed and produced by a WFO reaction
- This illustrates that one can compare objects directly
  - SQL only allows comparing attributes

Navigation for Query 7



Query 7

- List the molecules by code that are consumed and produced by a WFO reaction
- This is most easily handled by using a subquery

```
select m.code
  from Molecule m
 where exists(
         select *
           from Reaction r join r.consumes c
                join r.produces p
          where r.name='WFO'
            and c.molecule = m
            and p.molecule = m
       )
```

Query 8

- List all molecules by code that are catalysts. A catalyst is a molecule that is both consumed and produced by a reaction with the same counts.

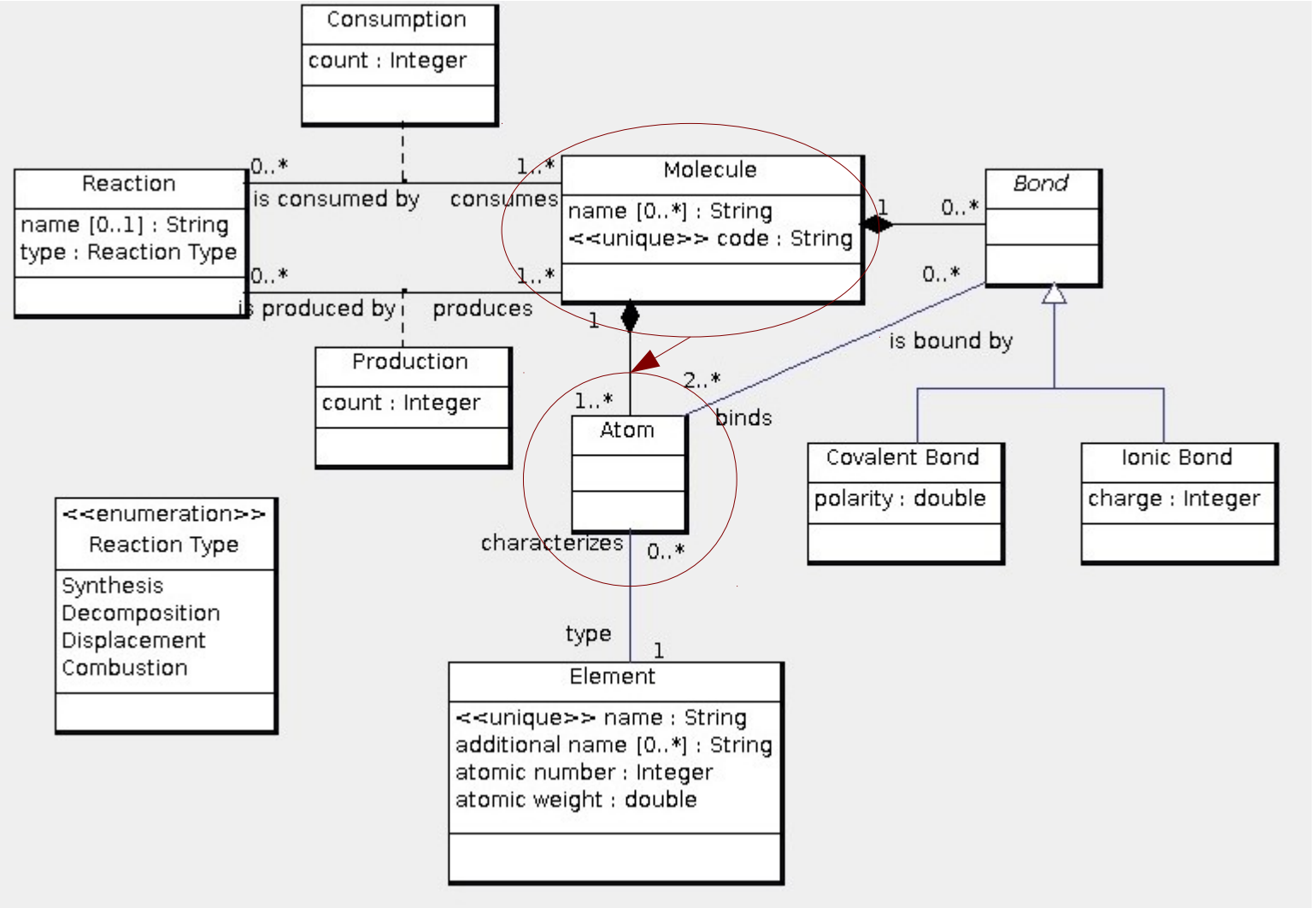
```
select m.code
  from Molecule m
 where exists(
         select *
           from Reaction r join r.consumes c
                join r.produces p
          where c.molecule = m
            and p.molecule = m
            and c.count = p.count
       )
```

Query 9

- Show the code and number of atoms for every Molecule

```
select m.code, count(*)
  from Molecule m join m.contains a
 group by m
```

Navigation for Query 9



Query 10

- Show the code and number of atoms for every Molecule that has at least 4 atoms

```
select m.code, count(*) ct
  from Molecule m join m.contains a
 group by m
having ct >= 4
```

Query 11

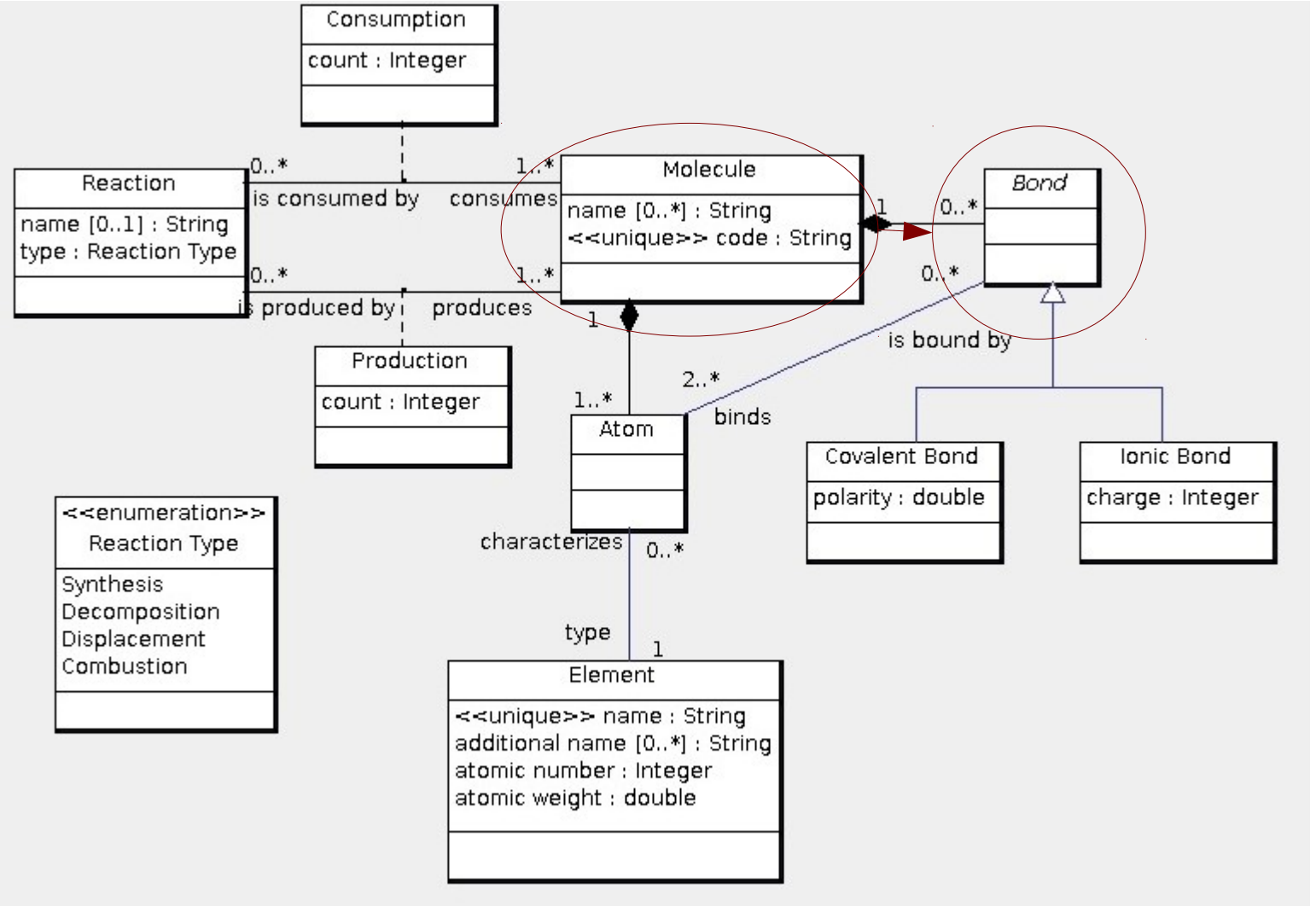
- Show the code and number of atoms in order by the number of atoms for every Molecule that has at least 4 atoms

```
select m.code, count(*) ct
  from Molecule m join m.contains a
 group by m
having ct >= 4
 order by ct
```

Query 12

- Show the code and number of bonds for every molecule
- Begin with the navigation...

Navigation for Query 12



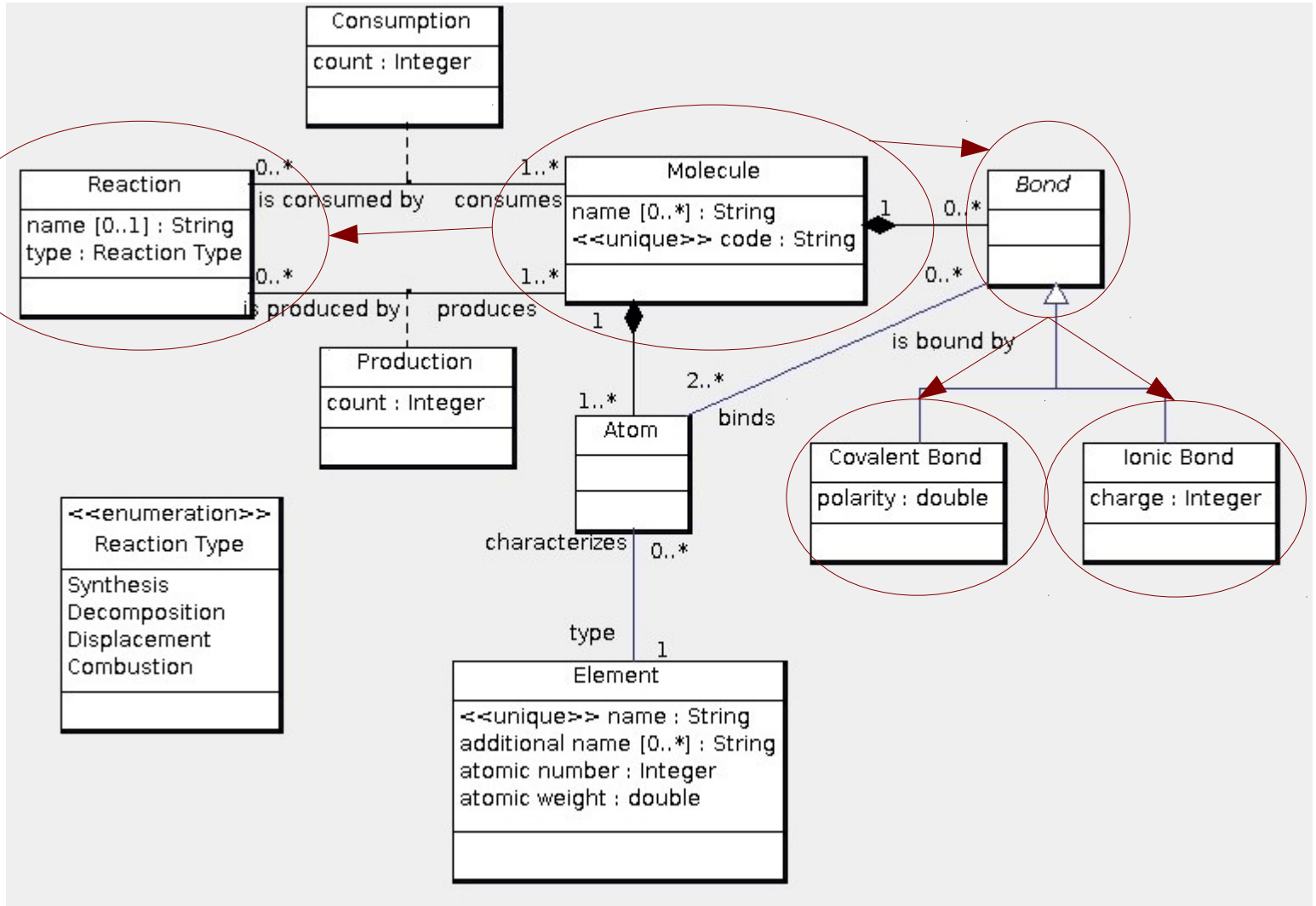
Query 12

- Show the code and number of bonds for every molecule
- This seems to be very similar to Query 9, but it is possible for a molecule to have no bonds at all.
- If a molecule has no bonds, then the join will have no row for that molecule so there will not be a group.
- An *outer join* can be used to solve this problem.

```
select m.code, count(b.id)
  from Molecule m left join m.contains b
 group by m
```

Query 14

- Show the code and bonds for every molecule produced by a combustion reaction, showing the charge of an ionic bond and polarity of a covalent bond. If a molecule has no bonds, then do not show it at all.
- Begin with the navigation...



Query 14

- Show the molecules and their bonds for molecules produced by a combustion reaction, showing the charge of an ionic bond and polarity of a covalent bond. If a molecule has no bonds, then do not show it at all.
- This will require a nested query as well as outer joins
  - To constrain the molecules to those produced by combustion reactions, use a nested query
  - To show the attributes of the subclasses, one just uses the attributes

Solution to Query 14

```
select m.id, b.charge, b.polarity
  from Molecule m join m.contains b
 where exists(
         select *
           from Reaction r join r.produces p
          where m = p.molecule
            and r.type = 'Combustion'
       )
```

Query 15

- What are the molecules by code that have a name?

```
select m.code
  from Molecule m
 where m.name.size() > 0
```

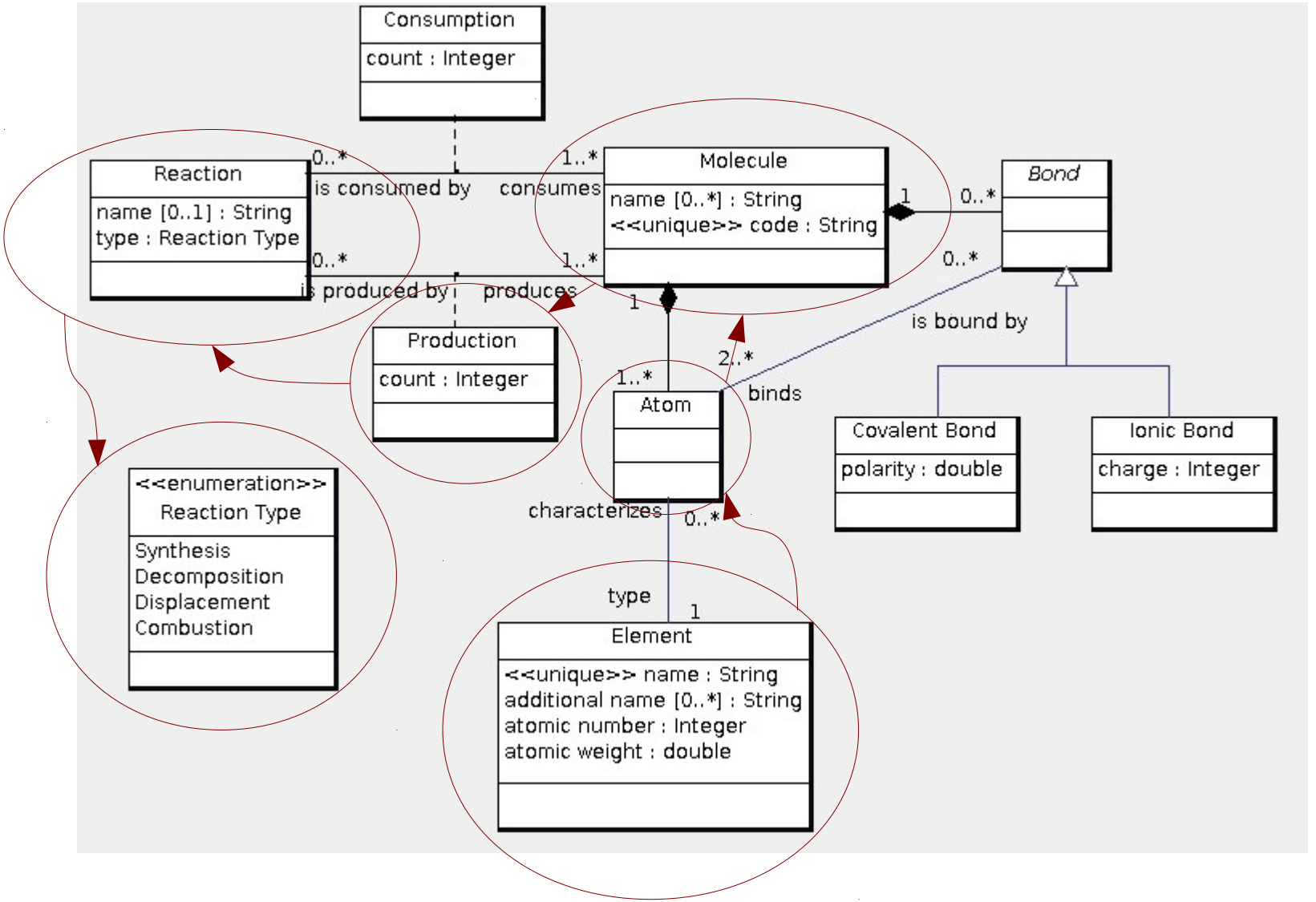
Query 16

- List all molecules by code that have a bond

```
select m.code
  from Molecule m
 where m.contains.size() > 0
```

Query 18

- List all elements, showing name, atomic number and atomic weight, that are produced in a synthesis reaction
- Start by navigating...



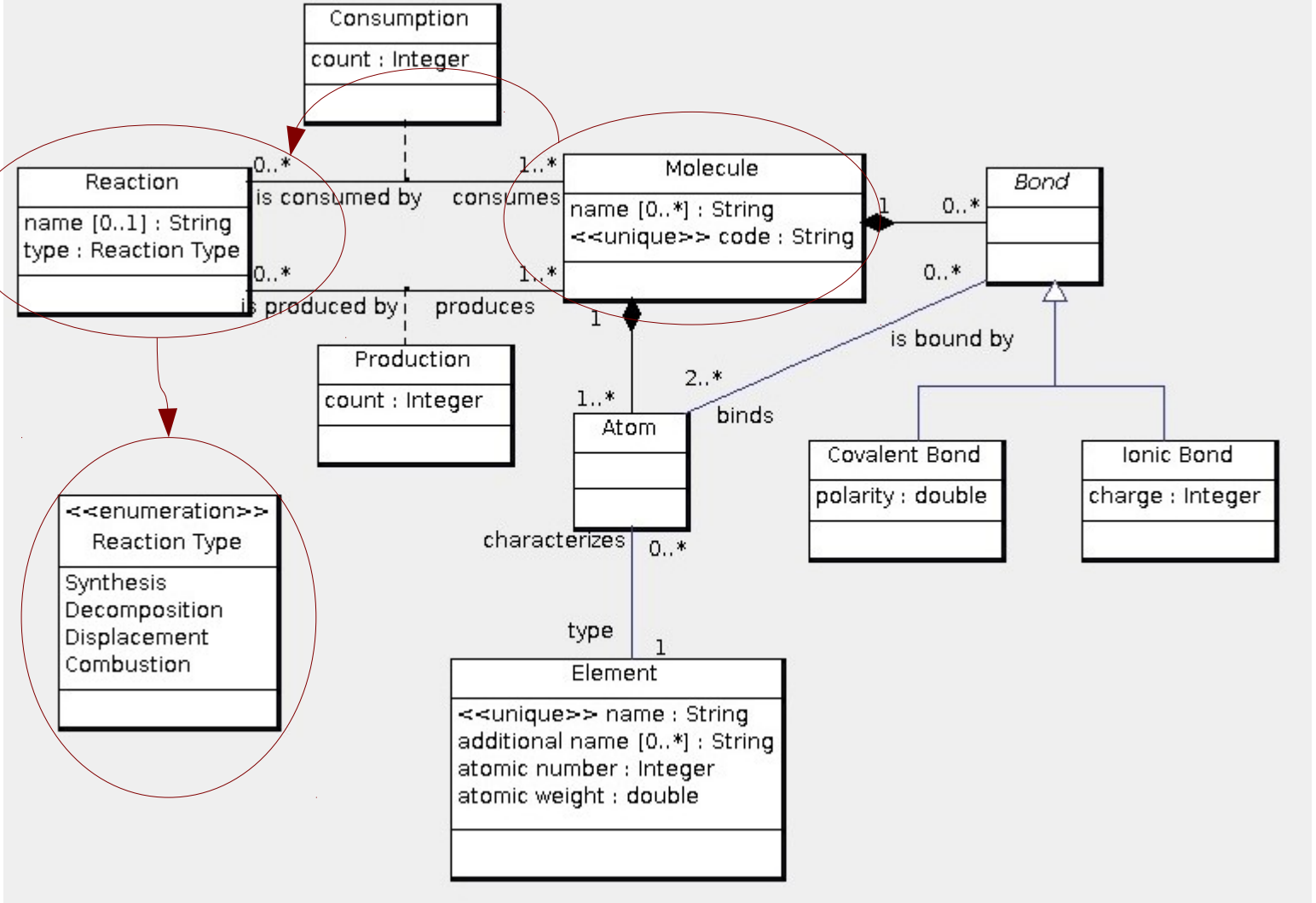
Query 18

- List all elements, showing name, atomic number and atomic weight, that are produced in a synthesis reaction

```
select e.name, e.atomicNumber, e.atomicWeight
  from Element e
 where exists(
         select *
           from Atom a join a.partOf.produces p
          where a.type = e
            and p.reaction.type = 'Synthesis'
       )
```

Query 19

- List all molecules by code that are consumed by a combustion reaction
- Start by navigating...



Query 19

- List all molecules by code that are consumed by a combustion reaction
- This should now be relatively routine:

```
select m.code
  from Molecule m
 where exists(
         select *
           from Reaction r join r.consumes c
          where c.consumes = m
            and r.type = 'Combustion'
       )
```

Query 20

- List all molecules by code that are consumed by every combustion reaction
- Rewrite this query as follows:
  1. List all molecules m by code such that for every combustion reaction r the molecule m is consumed by r
  2. List all molecules m by code such that there does not exist a combustion reaction r such that the molecule m is not consumed by r

Solution to Query 20

- List all molecules by code that are consumed by every combustion reaction

```
select m.code
  from Molecule m
 where not exists(
         select *
           from Reaction r
          where r.type='Combustion'
            and not exists(
                  select *
                    from Consumption c
                   where c.consumes = m
                     and c.isConsumedBy = r
                )
       )
```

Query 21

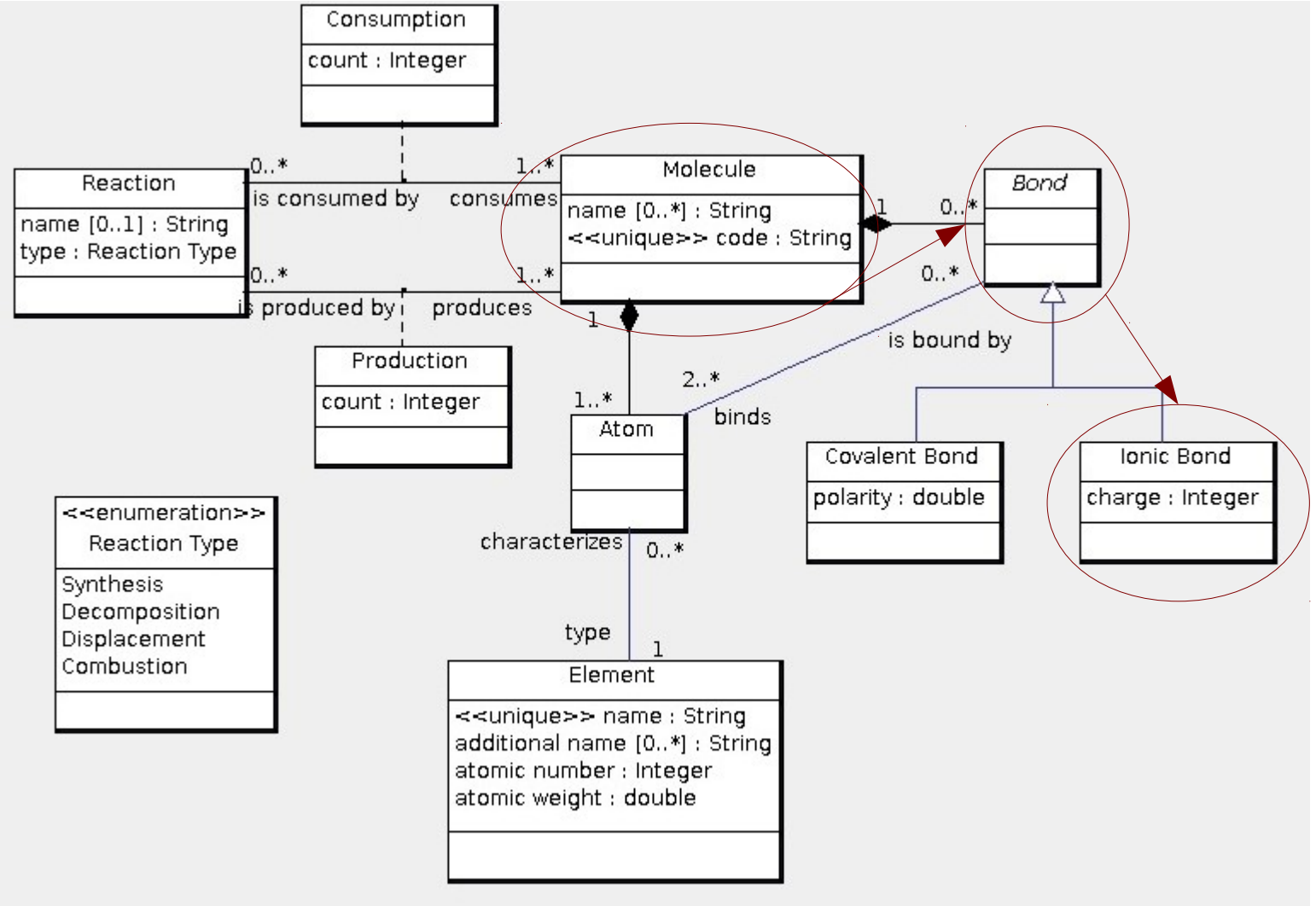
- List all elements, showing name, atomic number and atomic weight, that are produced by **every** synthesis reaction
- This looks like the same query as Query 18, but now **every** synthesis reaction rather than **some** synthesis reaction
- Rewrite the query like this:
  1. List all elements `e`, showing name, atomic number and atomic weight, such that for every synthesis reaction `r`, the element `e` is produced by `r`
  2. List all elements `e`, showing name, atomic number and atomic weight, such that there does not exist a synthesis reaction `r` such that the element `e` is not produced by `r`

Solution to Query 21

```
select e.name, e.atomicNumber, e.atomicWeight
  from Element e
 where not exists(
         select *
           from Reaction r
          where r.type = 'Synthesis'
            and not exists(
                  select *
                    from Atom a join
                         a.partOf.isProducedBy p
                   where a.type = e
                     and p.reaction = r
                )
       )
```

Query 22

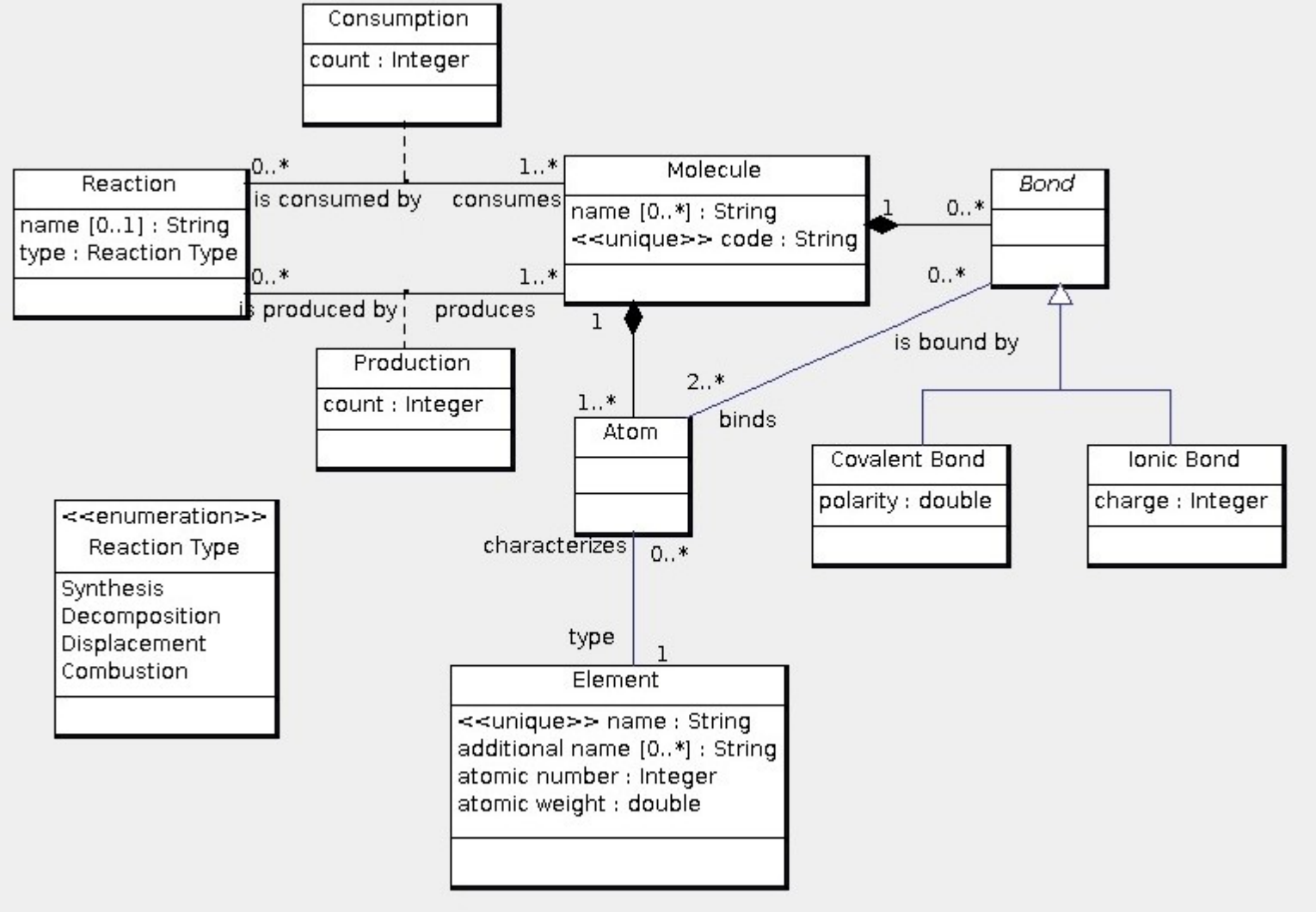
- List all molecules by code that have an ionic bond with charge at least 5
- Begin by navigating...



Solution to Query 22

- List all molecules by code that have an ionic bond with charge at least 5

```
select m.code
  from Molecule m
 where exists(
         select *
           from Bond b
          where b.partOf = m
            and type(b) = IonicBond
            and b.charge >= 5
       )
```



Number of Elements Query

- Find the number of elements in each reaction
- The result should consist of a set of pairs:
  - The reaction object
  - The number of elements that the reaction produces or consumes
- Atoms are preserved by reactions so one can use either produces or consumes for this query.
- Write your answer on a piece of paper.

Navigability

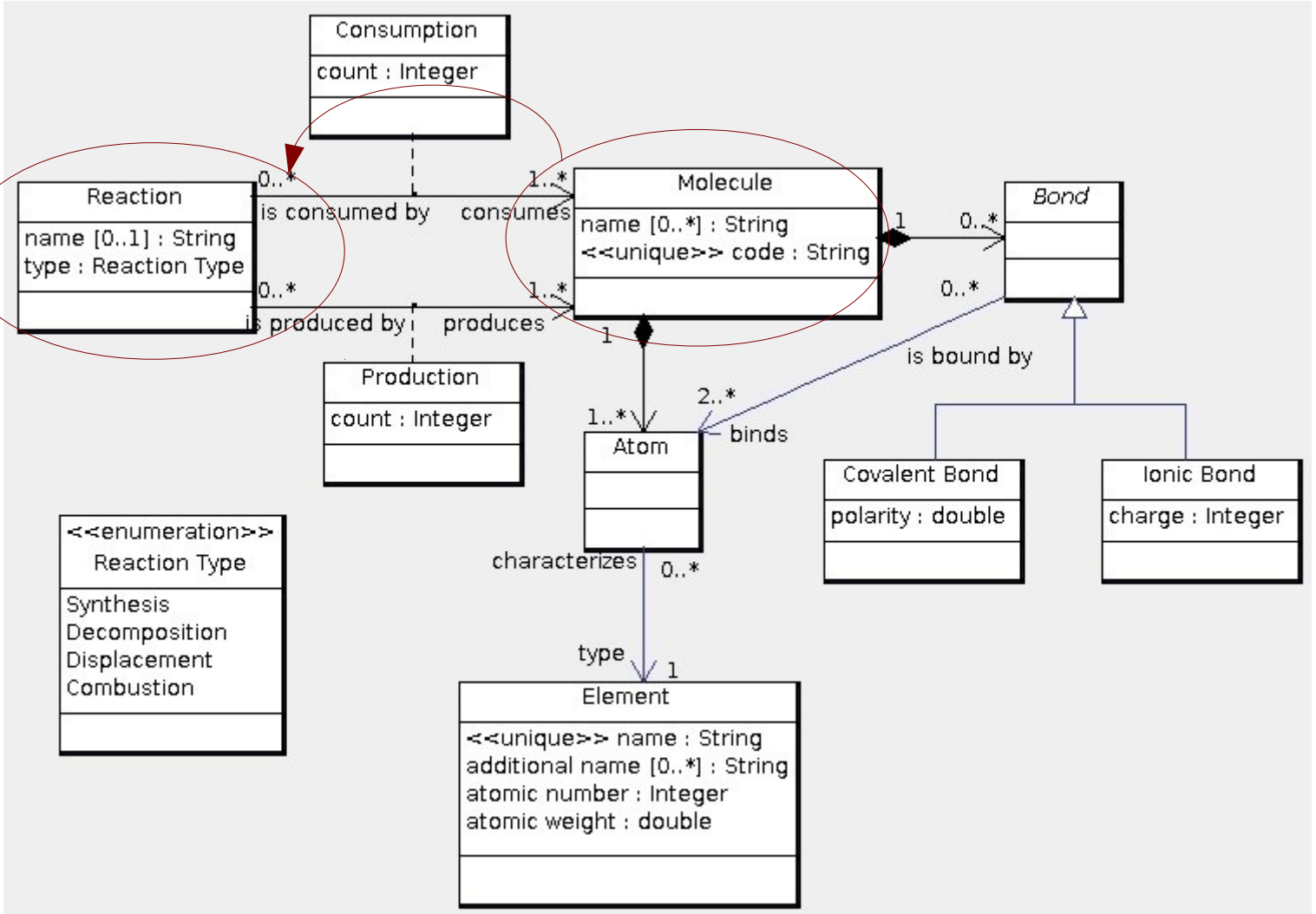
Impact of Navigability

- The queries done so far assume navigability in all the directions of associations that were used
- The following are some examples of how to deal with associations which are not bidirectional

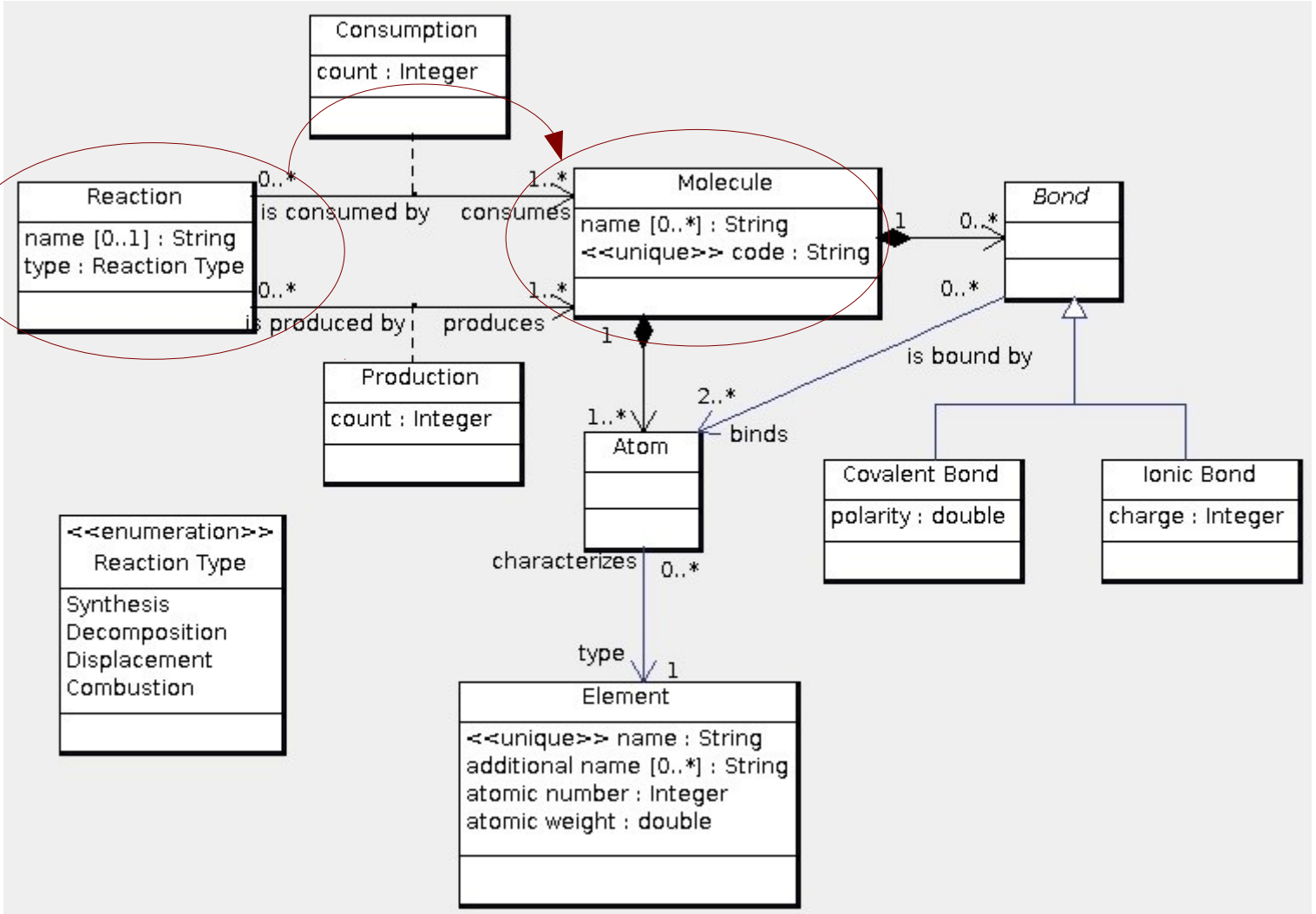
Query 6 Again

- List all molecules consumed by a reaction named WFO
- The original navigation cannot be used because the association is not navigable in that direction

Original Navigation for Query 6



New Navigation for Query 6



Query 6

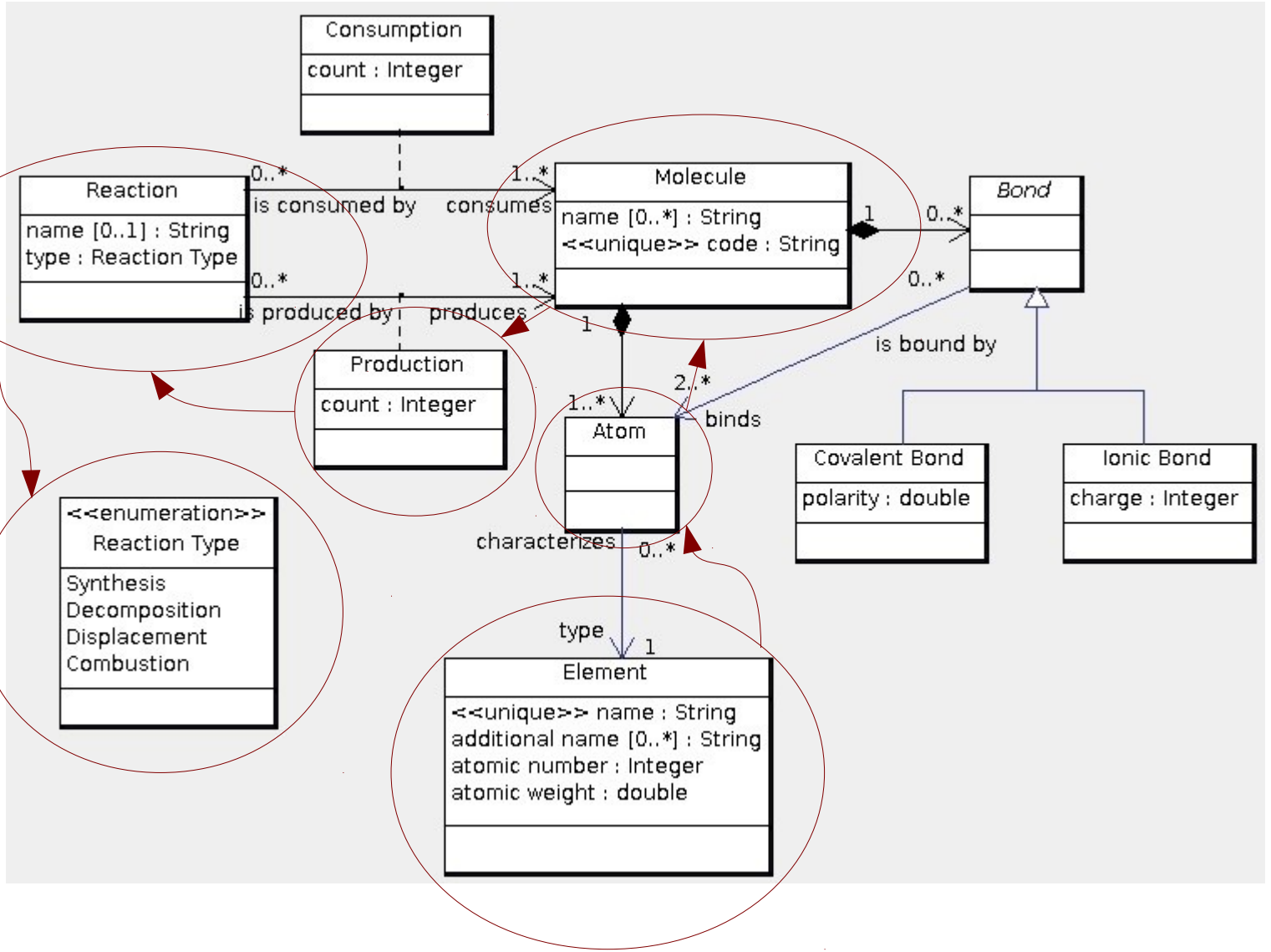
- The query navigates to the Consumption class from the Reaction class
  - This is one-to-many
- Then it navigates to the Molecule class
  - This is many-to-one
- The query is:

```
select c.molecule
  from Reaction r join r.consumes c
 where r.name='WFO'
```

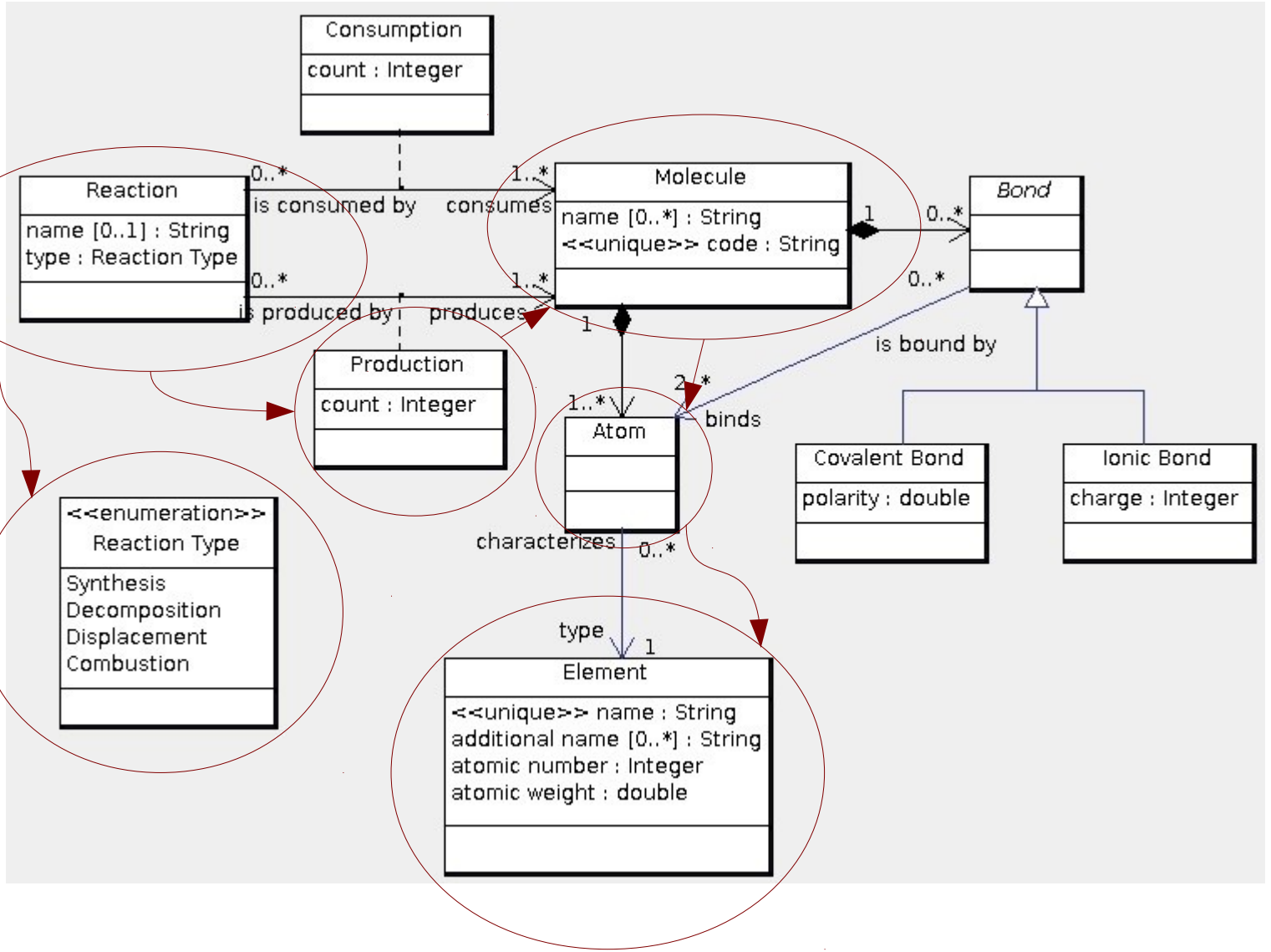
Query 18 Again

- List all elements, showing name, atomic number and atomic weight, that are produced in a synthesis reaction
- The original navigation is almost entirely in unsupported directions

Original Navigation for Query 18



New Navigation for Query 18



Query 18

- List all elements, showing name, atomic number and atomic weight, that are produced in a synthesis reaction

```
select e.name, e.atomicNumber, e.atomicWeight
  from Element e
 where exists(
         select *
           from Reaction r join r.isProducedBy p
                join p.molecule.binds a
          where a.type = e
            and r.type = 'Synthesis'
       )
```

Assignment #10

Final Exam Review

Final Exam Structure

- Same structure as on the Mid-Term Exam
  - Different topics
  - Longer
- Exam will primarily cover material used in assignments
- Questions will mostly be similar to the assignments

Logistics

- Monday, 24 April 2017
- 6:00PM to 8:00PM
- Shillman 105
- Laptops
  - Must have permission
  - Permission for Mid-Term is inherited by Final Exam
- Paper for writing your answers will be provided
  - You may use your own paper if desired
- Turn in only your answers

Topics

- Index Design
- Integrity
- Security
- Index structures
- Database programming
- Hierarchical queries
- Transactions
- Concurrency control
- Recoverability
- Metadata
- Object-relational mapping
- JPQL
- Storage devices

Index Design

- Given a set of queries and commands find the candidate indexes
- Only one design for the set, not one design per query
- Must include all required indexes
- Must specify type of every index
- If B-tree has multiple columns, must specify order of columns in the index

Index Design

- Index Types
  - Hash table
  - B-tree
- Required indexes
  - Primary key
  - Uniqueness constraints
  - Target of foreign key constraint
- Optional indexes
  - Selection
  - Join
- Index Design
  - Both required and optional indexes
  - Specify type for each index
  - Order of columns for B-tree indexes

Integrity

- Check constraints
- Assertions
- Triggers
  - Strategies for enforcement of constraints
- Stored procedures
- Dealing with lack of support for assertions

Security

- Granting privileges
  - Roles
  - Delegating privileges
- Revoking privileges
- Types of privilege
- Views for granting privileges
  - Only necessary for parts of a table

Index Structures

- Hash tables
- B-trees
- Comparison
  - B-trees support both exact match and range queries
  - Hash tables support only exact match queries
  - Hash tables are faster, less complicated
- Content Management

Database Programming

- Drivers
- Connections
- Prepared statements
- Result sets
- Errors and warnings
- Cursors
  - Not on final exam
- Times and dates
- Large objects
- Generated values
  - autoincrement
- Fill-in missing code
  - Do not rewrite given code
  - Do not give multiple answers
  - Only show your answer, not the given code

Hierarchical Queries

- Hierarchical model
  - Infosets
- Hierarchical schemas
  - DTD
  - XML Schema
- Query languages
  - Analogies with SQL
- XPath and HAPL
  - axes
  - node tests
  - predicates
  - functions
  - unions

Transactions and Concurrency Control

- Transaction concept
  - ACID properties
- Schedules
- Equivalence of schedules
  - View
  - Conflict
- Serial schedules
- Serializability: View, Conflict
- Concurrency control
  - Two-phase locking
- Recoverability concepts
  - Recoverability
  - Cascadelessness
  - Strictness
- Isolation levels
  - Phantoms

Metadata

- Relational metadata
  - JDBC API
- Content management metadata
- UML and MOF

Object-Relational Mapping

- Annotations
- Navigation
- Access to fields
- Fetching fields
- Customization files
- Synchronization
  - Transient
  - Managed
  - Removed
  - Detached
- JPQL

Storage Devices

- Hard disk drives
- Solid state drives
- Tapes
- Comparison of storage devices

Database Internals

- Buffer manager
- Log manager
- UNDO and REDO rules
  - Write Ahead Log Rule
- Lock manager
- Transaction manager
- Recovery manager
- Checkpoints
- Fuzzy backups
- Media failure recovery
- Total loss recovery
  - Hot standby
- Query optimization