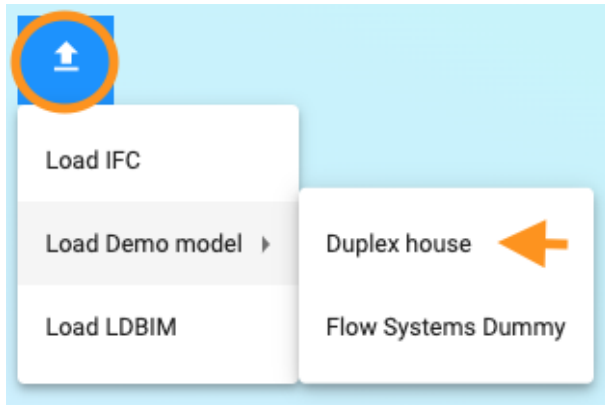


Queries on the Duplex house

In this assignment, we will be doing some queries in LD-BIM (<https://ld-bim.web.app>). We will use the Duplex House demo model.



SPARQL 101 ([Cheat Sheet](#))

Let's first do a simple query where we ask for all instances of [bot:Space](#).

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bot: <https://w3id.org/bot#>

SELECT ?space
WHERE{
  ?space rdf:type bot:Space .
}
```

Have a look at the query above. We can divide it into three areas of interest:

- First, we have a set of prefixes. A prefix is an abbreviation used in the [Turtle](#) RDF serialization to simplify query writing. Remember that everything in Linked Data is identified with HTTP URIs i.e. web addresses? Providing the information *PREFIX bot: <https://w3id.org/bot#>* tells the interpreter that every time it sees *bot:* it is an abbreviation for that URI. *bot:Space* is therefore interpreted as *<https://w3id.org/bot#Space>*. It is also possible to write the full URIs in *<triangle brackets>*, so the below query will yield the same results.

```
SELECT ?space
WHERE{
  ?space <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <https://w3id.org/bot#Space> .
}
```

- Next part is the *SELECT* clause which defines what we wish to return. *SELECT* will return a table with all the variables listed. There are also other clauses like *CONSTRUCT*, *ASK*, *INSERT* and *DELETE*.
- The last part is the *WHERE* clause. Here we define a triple pattern that should be matched. Anything prefixed with a ? (questionmark) indicates a variable,

so the triple pattern “*?anySubject ?anyRelationship ?anyObject*” would return everything in the graph since we only provide variables. This is typically not what we want, so the result can be restricted by replacing one of the three variables with a constant. In the above example, the relationship is restricted to *rdf:type* (what relates an instance with its type) and the object is limited to *bot:Space*. Therefore, *?space* will bind to all instances of *bot:Space* in the graph.

When we execute the query in LD-BIM we get a list of space URIs prefixed with *inst:* which in LD-BIM is used for all triples in the instance namespace. Try clicking the eye icon that shows up next to the space column header when hovering. This will highlight all spaces in random colours as shown in the image below. Now toggle the “Append new” setting, hover a row in the results list and click the eye at an individual space. This will highlight that single space. Click another space and that space will be highlighted. Toggle “Append new” on and as you continue to click spaces, they are appended to the scene. You can also click on an empty place in the scene to reset colors.



The screenshot shows the LD-BIM interface. At the top, a 3D model of a building is displayed with various colored blocks representing different spaces. Below the model, the interface shows the query results. On the left, the query is defined as follows:

```

1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX bot: <https://w3id.org/bot#>
3
4 SELECT ?space
5 WHERE{
6   ?space rdf:type bot:Space .
7 }
8

```

On the right, the results table shows 21 completed results. The first row is highlighted, and an eye icon is visible next to the space column header. The results are as follows:

space	inst:2gRXFgjRn2HPE%24YoDLX3FC
inst:OBTFw6f90Nfh9rP1dL3Q	
inst:2gRXFgjRn2HPE%24YoDLX3FC	
inst:OBTFw6f90Nfh9rP1dLXrc	
inst:OBTFw6f90Nfh9rP1dL_CZ	
inst:OBTFw6f90Nfh9rP1dLXri	
inst:OBTFw6f90Nfh9rP1dL_3G	

Now let’s simplify the query a bit by replacing *rdf:type* with *a* like shown below. This is syntactic sugar in Turtle and SPARQL that means the same thing. The *bot* prefix is still necessary but throughout this assignment we leave out prefixes that have already been specified previously to save space.

```

SELECT ?space
WHERE{
  ?space a bot:Space .
}

```

Returning more variables

Let's remove one of the constants in the query and replace it with a variable so we can return more results.

```
SELECT ?something ?class
WHERE{
  ?something a ?class .
}
```

What we are basically saying here is that we wish to return anything and the class it belongs to. “a” is used to relate an instance to its class, so this makes sense. We explicitly state the names of the classes, but we could also just use an asterisk like shown below to return all results.

```
SELECT *
WHERE{
  ?something a ?class .
}
```

Extending the pattern

We are not reduced to only using single-line triple pattern matches. The dot at the end of the line indicates that the triple is over, and we are free to add another condition that should also be met. We could for example ask for all spaces including all relationships starting from a space and going to other objects. Notice that we must use the same variable name to bind to all the spaces that were retrieved from the first pattern match.

```
SELECT *
WHERE{
  ?space a bot:Space .
  ?space ?property ?value .
}
```

This way we can traverse the graph and discover new information. We also extend the SELECT clause to only return unique properties. This way we will know all the different relationships that exist on spaces in our dataset.

This is a useful query pattern that can be used at any step when defining your full query. It can be used at any point to ask the question “where can I get from here” since it reveals all outgoing relationships from the current position (in this case all outgoing relationships from any space).

```
SELECT DISTINCT ?property
WHERE{
  ?space a bot:Space .
  ?space ?property ?value .
}
```

Semicolon and comma

A dot indicates that a triple is over, but we can also use semicolon which means that the subject of the previous triple is still valid for the next triple. The below query is therefore equal to the one above.

```
SELECT DISTINCT ?property
WHERE{
  ?space a bot:Space ;
  ?property ?value .
}
```

Using a colon indicates that both the subject and the predicate should be repeated. For example, the query below will return all instance of bot:Element and will further return the other classes that the element belongs to.

```
SELECT *
WHERE{
  ?element a bot:Element , ?class .
}
```

Limiting results

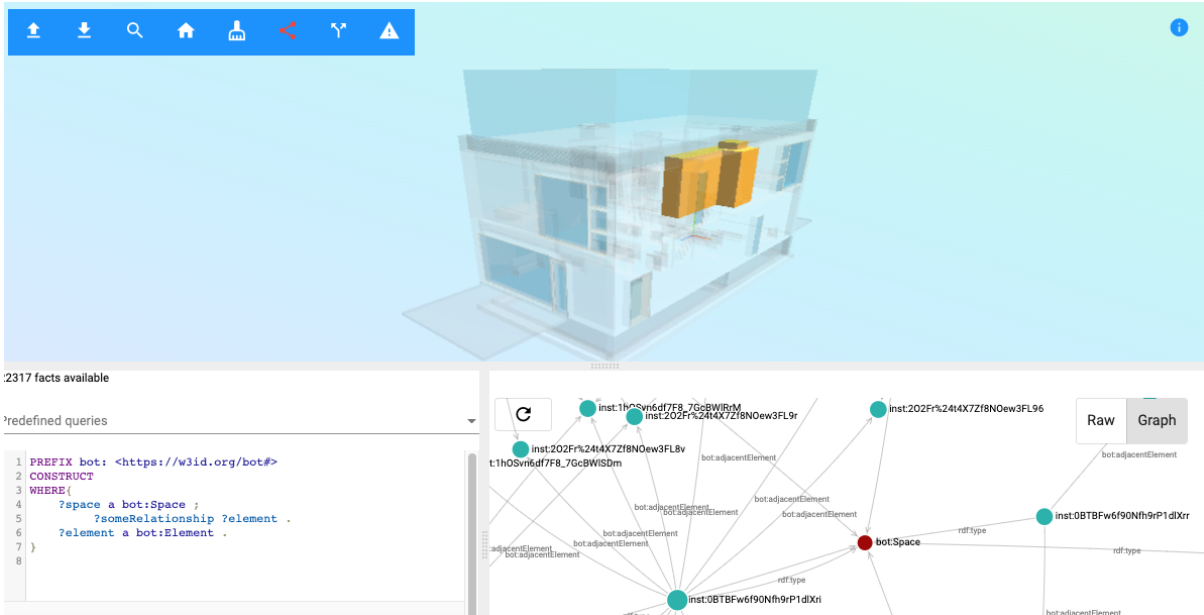
The above query will also bind the bot:Element class to the ?class variable since it also matches the pattern. We might, however, not be interested in this. We can use a filter to leave these out from the returned results.

```
SELECT *
WHERE{
  ?element a bot:Element , ?class .
  FILTER(?class != bot:Element)
}
```

Construct queries

A construct query returns the sub-graph in a graph-like data structure like Turtle or JSON-LD. The sub-graph contains only the triples that are matched by the WHERE clause. The query below will return all instances of bot:Space and all their relationships to all instances of bot:Element. Notice that LD-BIM shows the results in a graph.

```
CONSTRUCT
WHERE{
  ?space a bot:Space ;
  ?someRelationship ?element .
  ?element a bot:Element .
}
```



2317 facts available

redefined queries

```

1 PREFIX bot: <https://w3id.org/bot#>
2 CONSTRUCT
3 WHERE{
4   ?space a bot:Space ;
5   ?someRelationship ?element .
6   ?element a bot:Element .
7 }
8

```

When you hover a node in that graph that represents an object in the IFC model that element will be highlighted. You can also click the Raw button to get the results in JSON-LD serialized RDF.

In the construct clause itself it is also possible to limit the results. We could for example omit the relationships to the classes and only return the instances and their relationships. When you use such advanced features in the WHERE clause it is no longer possible to just write CONSTRUCT. You also need to explicitly specify what should be returned.

```

CONSTRUCT{
  ?space ?someRelationship ?element .
}
WHERE{
  ?space a bot:Space ;
  ?someRelationship ?element .
  ?element a bot:Element .
}

```

Aggregation functions

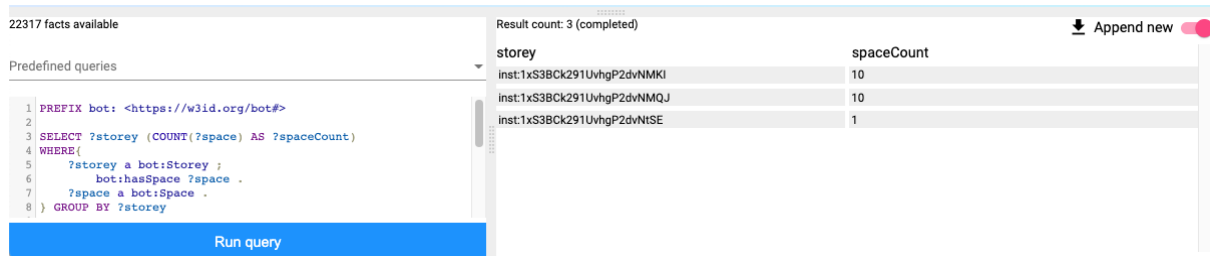
Let's run a simple aggregation function that counts the number of spaces in the graph. What we would like to find is the number of spaces per storey. We do this by querying relationships between storeys and spaces and grouping them by storey. In the SELECT clause we use the aggregate function COUNT() to count the number of space occurrences per storey and finally we bind the result to a new variable ?spaceCount.

```

SELECT ?storey (COUNT(?space) AS ?spaceCount)
WHERE{
  ?storey a bot:Storey ;
  bot:hasSpace ?space .
  ?space a bot:Space .
}
GROUP BY ?storey

```

At the current state LD-BIM uses a SPARQL implementation based on N3/Comunica which is not so stable so you might have to execute the query a few times. Try refreshing the page, load the model again and run the same query. That usually works!



22317 facts available

Result count: 3 (completed) Append new

Predefined queries

```

1 PREFIX bot: <https://w3id.org/bot#>
2
3 SELECT ?storey (COUNT(?space) AS ?spaceCount)
4 WHERE {
5   ?storey a bot:Storey ;
6   bot:hasSpace ?space .
7   ?space a bot:Space .
8 } GROUP BY ?storey
  
```

Run query

storey	spaceCount
inst:1xS3Bck291UvhgP2dvNMKI	10
inst:1xS3Bck291UvhgP2dvNMQJ	10
inst:1xS3Bck291UvhgP2dvNISE	1

GROUP_CONCAT is another aggregate query that can be used to return all the individual spaces at each group. Again, the implementation is not strong in these advanced query patterns. It will likely change in near future since we are considering moving to Oxigraph which is also faster (see [preliminary test results](#)).

```


SELECT ?storey (GROUP_CONCAT(?space) AS ?spaces)
WHERE {
  ?storey a bot:Storey ;
  bot:hasSpace ?space .
  ?space a bot:Space .
} GROUP BY ?storey
  
```

Querying for space's adjacent elements seems to be more stable and the concept is the same. So if you can't get the above query to work, try this:

```

SELECT ?space (GROUP_CONCAT(?adjEl) AS ?adjacent)
WHERE {
  ?space a bot:Space ;
  bot:adjacentElement ?adjEl
} GROUP BY ?space ?area
  
```

LD-BIM understands that the adjacent-column contains a list of elements, so try first clicking the eye to color a space in the scene and then afterwards click the eye in the adjacent column on the same row. Try that with a few spaces and toggle "Append new" in-between to remove the previous results from the scene.



22317 facts available

Result count: 17 (completed) Append new

Predefined queries

```

1 PREFIX bot: <https://w3id.org/bot#>
2
3 SELECT ?space (GROUP_CONCAT(?adjEl) AS ?adjacent)
4 WHERE {
5   ?space a bot:Space ;
6   bot:adjacentElement ?adjEl
7 } GROUP BY ?space ?area
  
```

space	adjacent
inst:0BTBFw6f90Nfh9rP1dIXri	inst:1hOSvndf7F8_7GcBWIRm inst:202Fr%244X7Zf8N0ew3FL9r inst:202Fr%244X7Zf8N0ew3FL9r inst:202Fr%244X7Zf8N0ew3FL96 inst:20Brcmk58NupXoVOHUVV inst:202Fr%244X7Zf8N0ew3FL8v inst:1hOSvndf7F8_7GcBWISDm inst:20Brcmk58NupXoVOHUVPL inst:1aj%24VJZFnZTexpZUBckPac inst:20Brcmk58NuoXoVdHUtSW

Well-known text (advanced)

LD-BIM can also visualize [Well-known Text geometries](#) like Points, linestrings and polygons from the result table. In this example, we are building quite a complex query.

We are looking at a particular space that we know exists in the Duplex house (*inst:0BTBFw6f90Nfh9rP1dl_3A*) and we use a BIND clause to bind this space URI to a variable `?space`. This approach is clean since it puts the variable in the very top of the query making it easier to understand. If you wish to do the same with another space, simply replace the URI.

We are then looking for space boundaries, which are in LD-BIM modeled as instances of [bot:Interface](#). An interface is related to the interfacing objects that are interfacing using the [bot:interfaceOf](#) relationship. We are looking for relationships to spaces and elements, so we bind to representative variables. For the interface, we are also interested in the 3D vertices and we would like to restrict the results to only vertical space boundaries. We further restrict the element variable to only hold instances of [bot:Element](#).

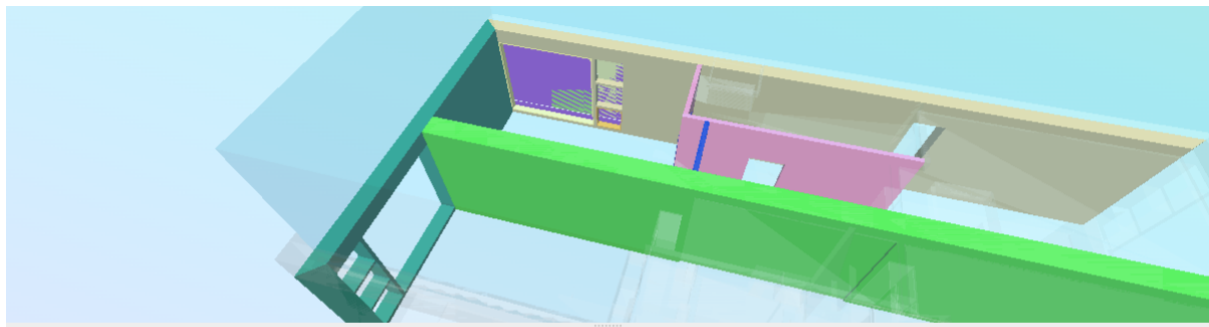
The last thing we do is that we use a BIND clause to build our WKT text strings in the form `POLYGON Z(x1 y1 z1, xn yn zn)` and bind them to the variable `?boundaryGeometry`.

```
PREFIX ex: <https://example.com/>
PREFIX kg: <https://w3id.org/kobl/geometry#>
PREFIX bot: <https://w3id.org/bot#>
PREFIX inst: <https://web-bim/resources/>

SELECT DISTINCT ?boundary ?boundaryGeometry ?element
WHERE{
  BIND(inst:0BTBFw6f90Nfh9rP1dl_3A AS ?space)

  ?boundary bot:interfaceOf ?space , ?element ;
    kg:vertices3D ?ver ;
    ex:isVertical true .
  ?element a bot:Element .
  BIND(CONCAT("POLYGON Z (", STR( ?ver ), ")") AS ?boundaryGeometry )
}
```

When you have executed the query, try first showing the full element column by clicking the eye.



22317 facts available

Result count: 11 (completed) Append new

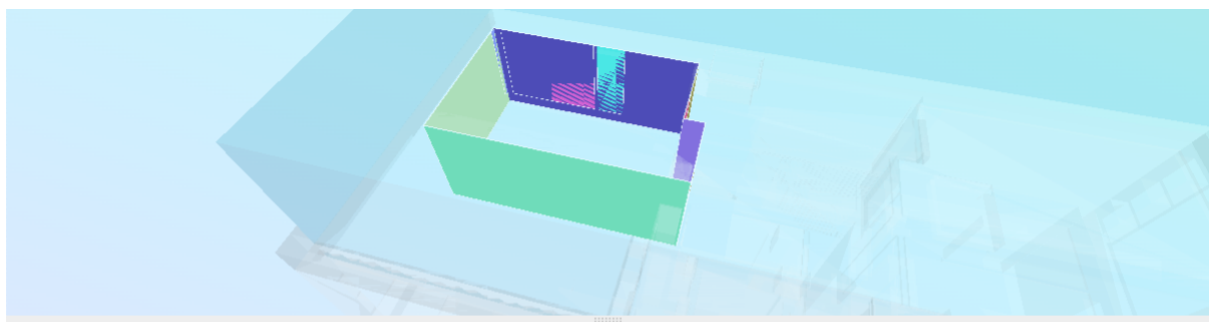
boundaryGeometry	element	boundary
POLYGON Z (0.208 14.583 3.428, 0.208 17.383 3.428, 0.208 17.383 5.699999999999999, 0.208 14.583 5.699999999999999, 0.208 14.583 3.428)	inst:10GAJRTFv8%24zmKJOH4%24e	inst:03zEL46e54kwMort0K0hk
POLYGON Z (2.444 11.072 3.1, 2.444 11.612 3.1, 2.444 11.612 5.7, 2.444 11.072 5.7, 2.444 11.072 3.1)	inst:202Fr%24t4X7Zf8N0ew3FLIE	inst:07La4RUKb3ve2KmqyWv7I
POLYGON Z (2.444 11.612 3.1, 0.208 11.612 3.1, 0.208 11.612 5.7, 2.444 11.072 5.7, 2.444 11.072 3.1)	inst:202Fr%24t4X7Zf8N0ew3FLJd	inst:00QK7TimCK9PekLFkbtZW

```

6 SELECT DISTINCT ?boundary ?boundaryGeometry ?element
7 WHERE{
8   BIND(inst:0BTBFw6f90Nfh9rP1dl_3A AS ?space)
9
10  ?boundary bot:interfaceOf ?space , ?element ;
11    kg:vertices3D ?ver ;
12    ex:isVertical true .
13  ?element a bot:Element .
14  BIND(CONCAT("POLYGON Z (", STR( ?ver ), ")") AS ?
    boundaryGeometry )

```

You will see all elements that are adjacent to the space and you will probably understand why we need space boundaries if we wish to address only the part of the element that is interfacing with the space. Now deselect “Append new” and show the boundaryGeometry column.



22317 facts available

Result count: 11 (completed) Append new

boundaryGeometry	element	boundary
POLYGON Z (0.208 14.583 3.428, 0.208 17.383 3.428, 0.208 17.383 5.699999999999999, 0.208 14.583 5.699999999999999, 0.208 14.583 3.428)	inst:10GAJRTFv8%24zmKJOH4%24e	inst:03zEL46e54kwMort0K0hk
POLYGON Z (2.444 11.072 3.1, 2.444 11.612 3.1, 2.444 11.612 5.7, 2.444 11.072 5.7, 2.444 11.072 3.1)	inst:202Fr%24t4X7Zf8N0ew3FLIE	inst:07La4RUKb3ve2KmqyWv7I

```

6 SELECT DISTINCT ?boundary ?boundaryGeometry ?element
7 WHERE{
8   BIND(inst:0BTBFw6f90Nfh9rP1dl_3A AS ?space)
9
10  ?boundary bot:interfaceOf ?space , ?element ;
11    kg:vertices3D ?ver ;
12    ex:isVertical true .
13  ?element a bot:Element .
14  BIND(CONCAT("POLYGON Z (", STR( ?ver ), ")") AS ?
    boundaryGeometry )

```

You can also click the individual elements and their related boundary one by one to investigate the results.

Now try replacing *inst:0BTBFw6f90Nfh9rP1dl_3A* with the URI of another space and see the results.

Lastly, try changing the “POLYGON Z” part of the BIND clause to “LINESTRING Z” to watch the results as linestrings instead of surfaces.

Try yourself

With what you have now learned, try formulating some queries on your own. Remember the trick you learned where you ask for outgoing relationships? Try that! You can also try clicking the “query resource” button in a cell like one of the space boundaries for example to create a query that will return everything we know about that resource. You can also try the Flow Systems demo model that contains different relationships like “Pipe supplies fluid to Fitting” and “System has component Pipe”.

Thoughts and perspectives

What you have been querying is the information that we have decided to extract from the IFC-file. It is also possible to get a full conversion of the IFC to [ifcOWL](#), but this data structure is a 1-1 representation of how it is modeled in the IFC which is really complicated to query. We just didn’t want to scare you away! Also, the LBD community is arguing that this complex representation is way too complex and carries on too much legacy, so after all maybe we don’t even need it in the future? LBD uses a modular approach where different domains can model their individual perspectives and bring them together as a federated whole and one might argue that this scales better.

You might have stumbled upon the fact that all properties are put in the instance namespace. This was a modeling decision made when building the IFC parser for properties. It would make sense to use the IFC namespace for all properties that are from PSets issued by BuildingSMART and then the rest of them could be described in the instance namespace.