CityJSON in combination with MongoDB, PostgreSQL and GraphQL

Karin Staring - 4952510

P5 presentation

Stelios Vitalis

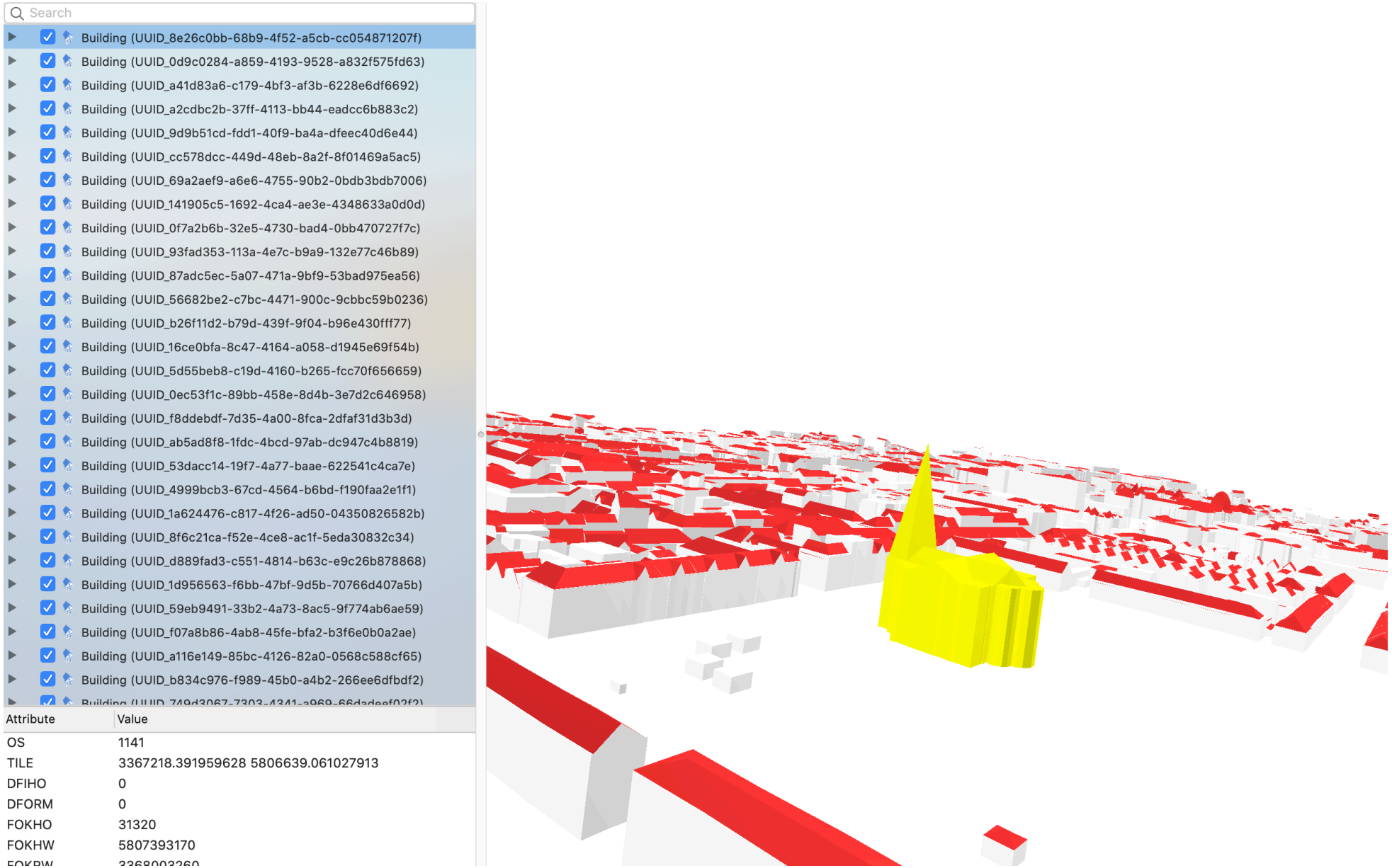
Linda van den Brink

Balázs Dukai

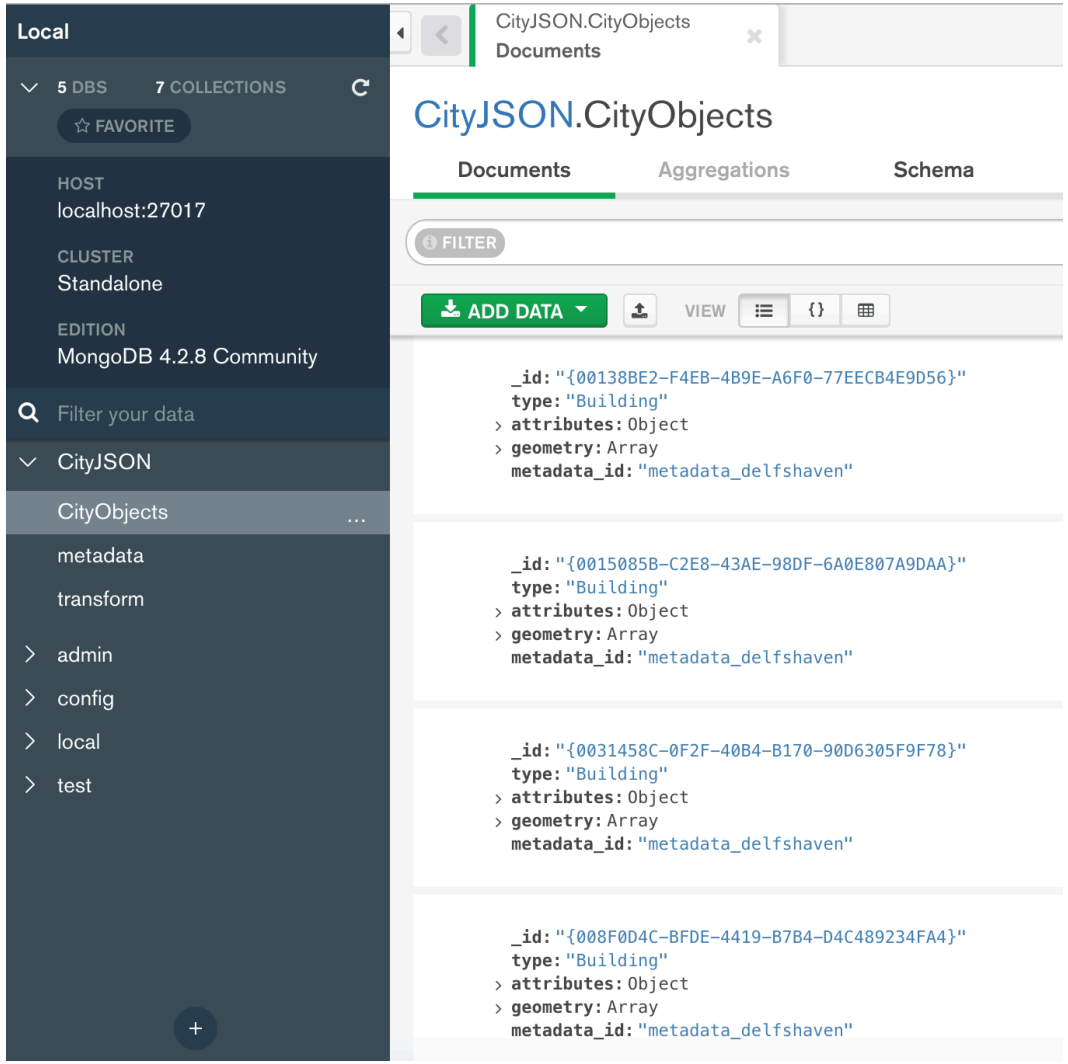
Willem Korthals Altes

TU Delft GEONOVUM

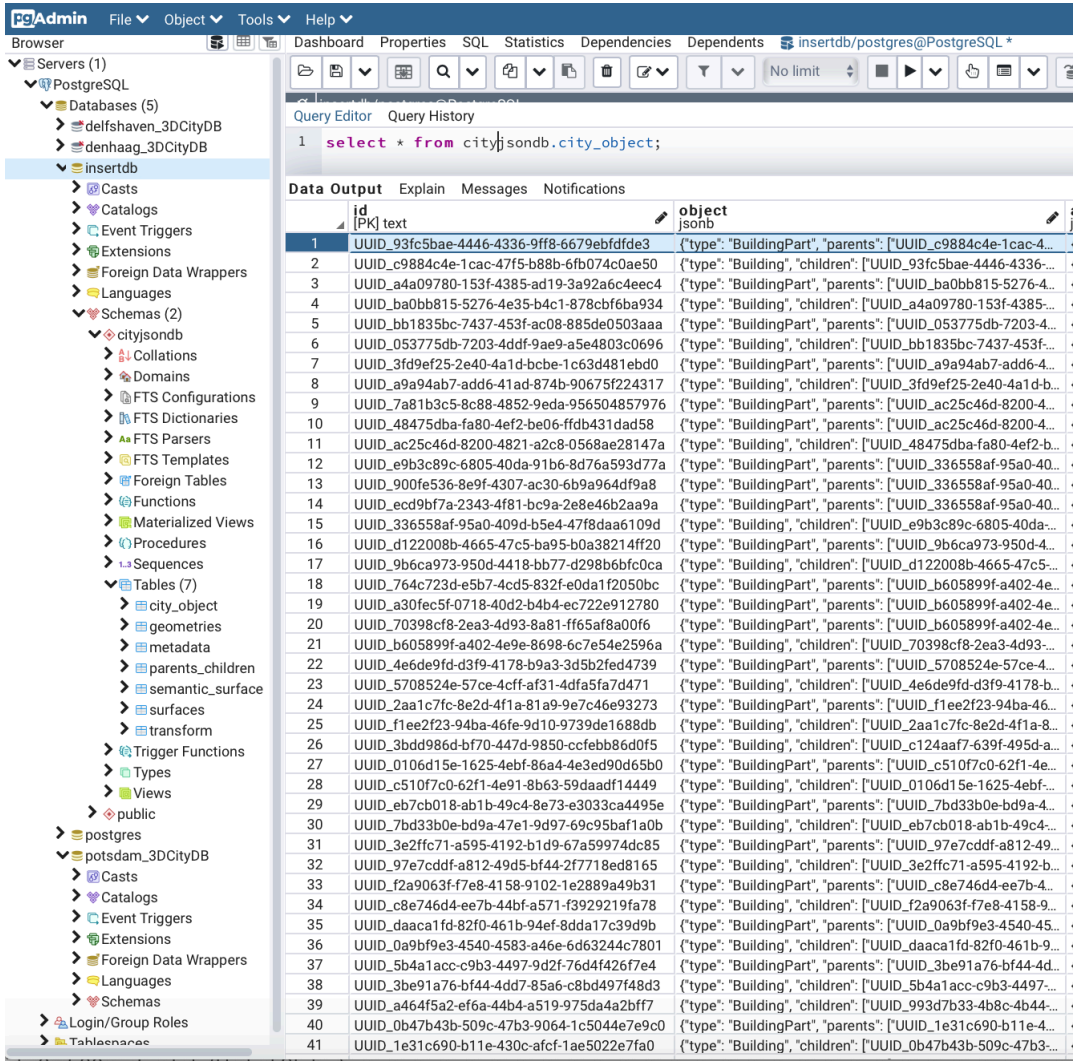

CityGML (XML)
CityJSON (JSON)


Search
Building (UUID_8e26c0bb-68b9-4f52-a5cb-cc054871207f)
Building (UUID_0d9c0284-a859-4193-9528-a832f575fd63)
Building (UUID_a41d83a6-c179-4bf3-af3b-6228e6df6692)
Building (UUID_a2cdbc2b-37ff-4113-bb44-eadcc6b883c2)
Building (UUID_9d9b51cd-fdd1-40f9-ba4a-dfeec40d6e44)
Building (UUID_cc578dcc-449d-48eb-8a2f-8f01469a5ac5)
Building (UUID_69a2aef9-a6e6-4755-90b2-0bdb3bdb7006)
Building (UUID_141905c5-1692-4ca4-ae3e-4348633a0d0d)
Building (UUID_0f7a2b6b-32e5-4730-bad4-0bb470727f7c)
Building (UUID_93fad353-113a-4e7c-b9a9-132e77c46b89)
Building (UUID_87adc5ec-5a07-471a-9bf9-53bad975ea56)
Building (UUID_56682be2-c7bc-4471-900c-9cbbc59b0236)
Building (UUID_b26f11d2-b79d-439f-9f04-b96e430fff77)
Building (UUID_16ce0bfa-8c47-4164-a058-d1945e69f54b)
Building (UUID_5d55beb8-c19d-4160-b265-fcc70f656659)
Building (UUID_0ec53f1c-89bb-458e-8d4b-3e7d2c646958)
Building (UUID_f8ddebdf-7d35-4a00-8fca-2dfaf31d3b3d)
Building (UUID_ab5ad8f8-1fdc-4bcd-97ab-dc947c4b8819)
Building (UUID_53dacc14-19f7-4a77-baae-622541c4ca7e)
Building (UUID_4999bcb3-67cd-4564-b6bd-f190faa2e1f1)
Building (UUID_1a624476-c817-4f26-ad50-04350826582b)
Building (UUID_8f6c21ca-f52e-4ce8-ac1f-5eda30832c34)
Building (UUID_d889fad3-c551-4814-b63c-e9c26b878868)
Building (UUID_1d956563-f6bb-47bf-9d5b-70766d407a5b)
Building (UUID_59eb9491-33b2-4a73-8ac5-9f774ab6ae59)
Building (UUID_f07a8b86-4ab8-45fe-bfa2-b3f6e0b0a2ae)
Building (UUID_a116e149-85bc-4126-82a0-0568c588cf65)
Building (UUID_b834c976-f989-45b0-a4b2-266ee6dfbdf2)
Attribute
Value
OS
1141
TILE
3367218.391959628 5806639.061027913
DFIHO
0
DFORM
0
FOKHO
31320
FOKHW
5807393170

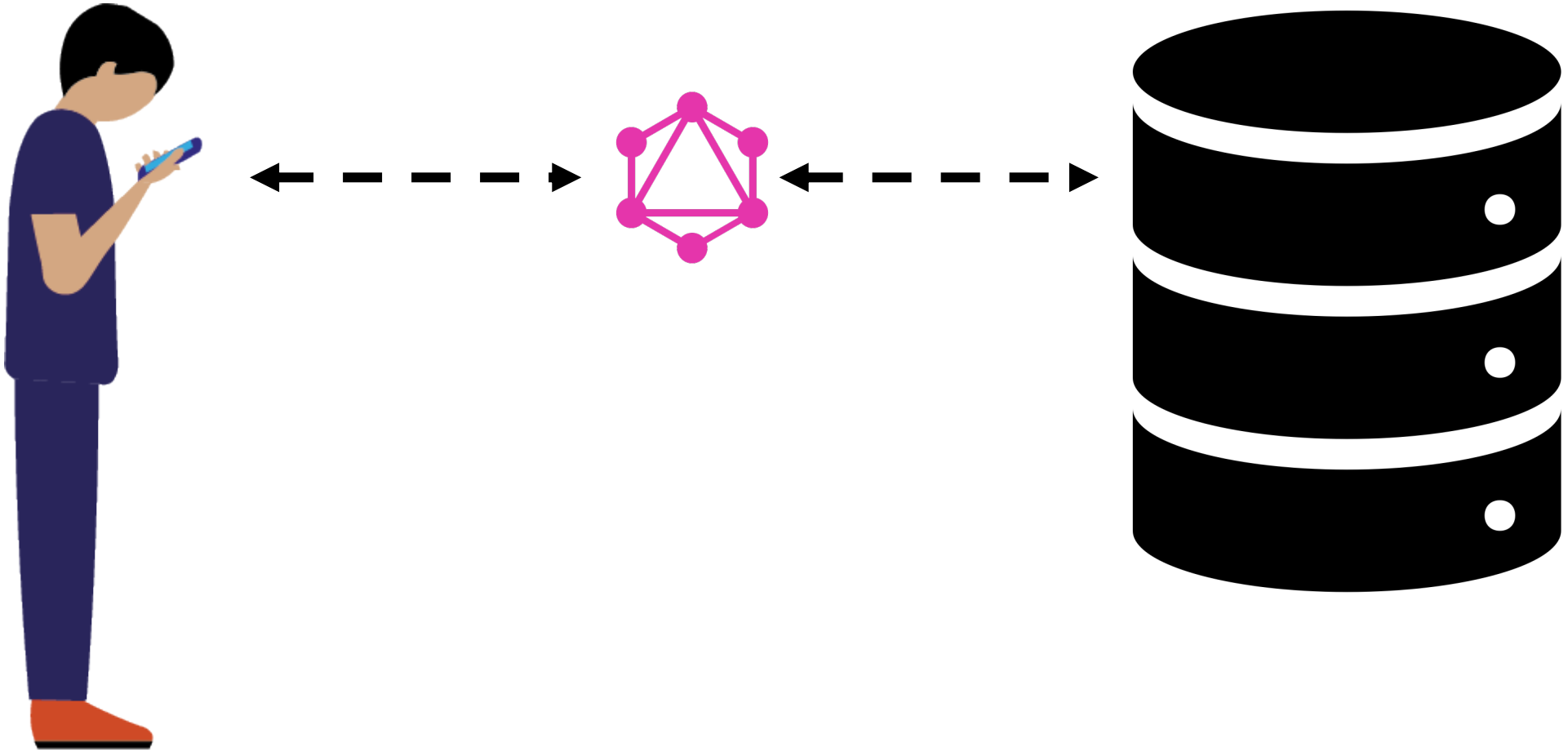
MongoDB = NoSQL/document database



PostgreSQL = Relational database



Use case

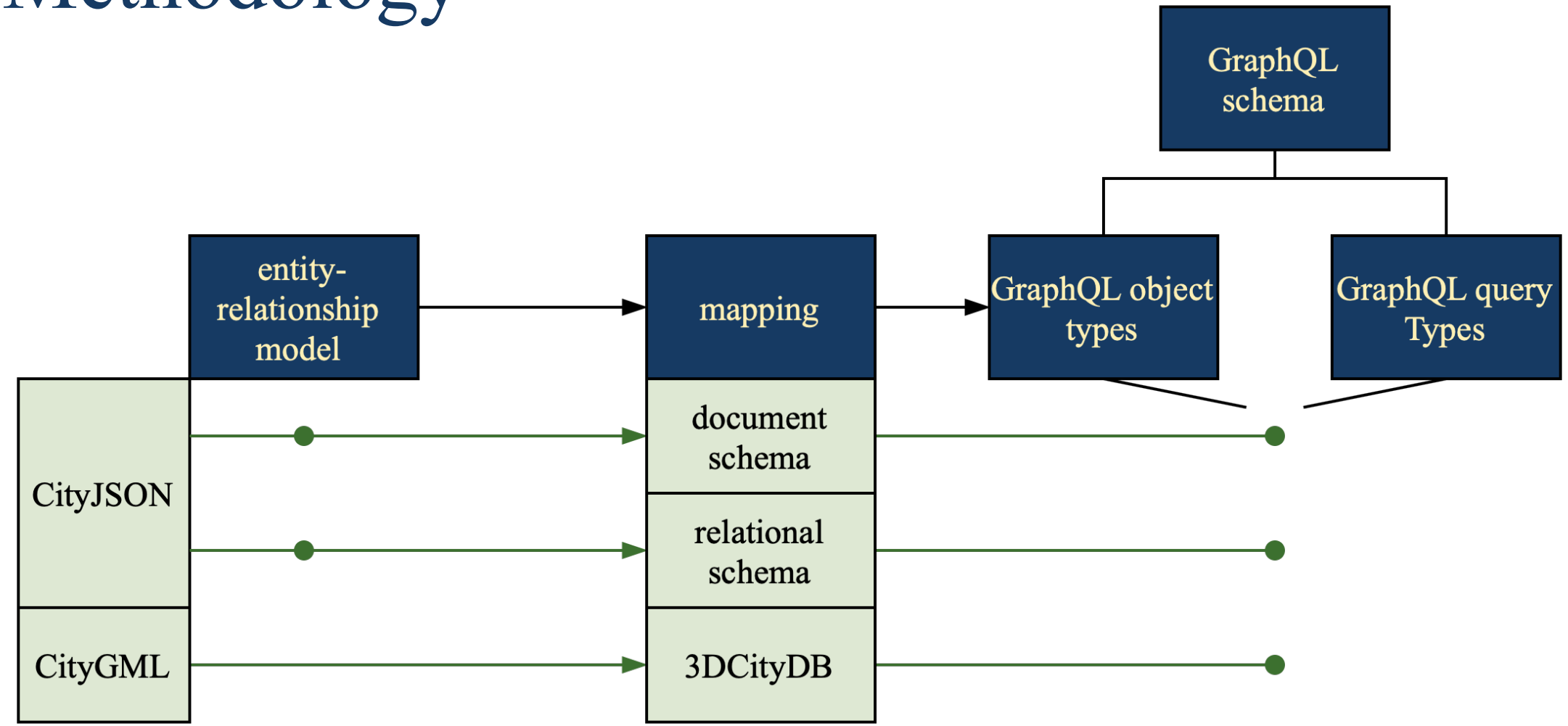


Research

How suitable are MongoDB and PostgreSQL for the storage and querying of CityJSON using GraphQL?

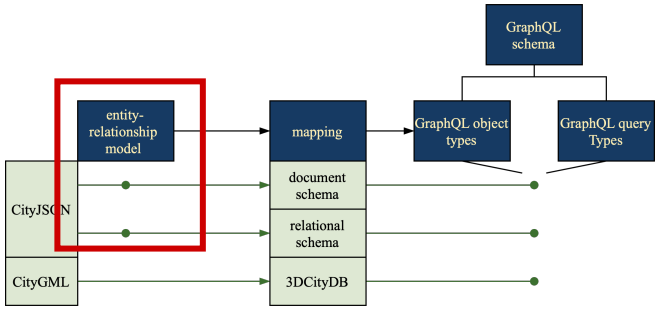
TU Delft

Methodology

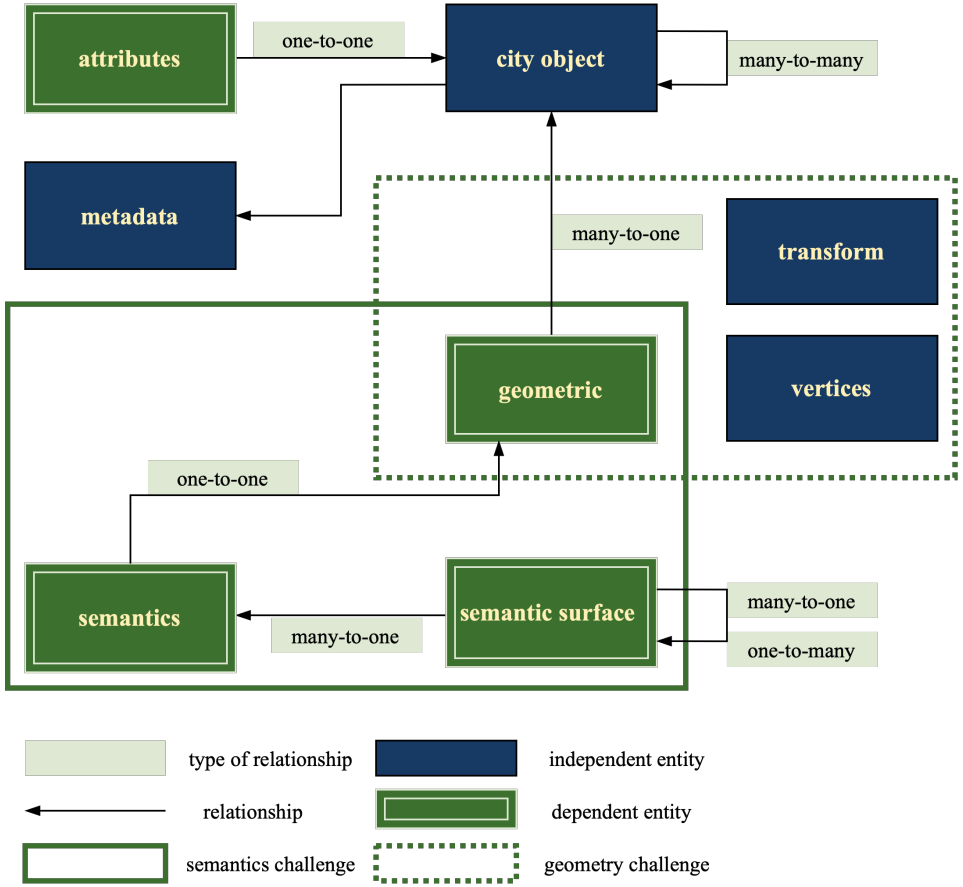


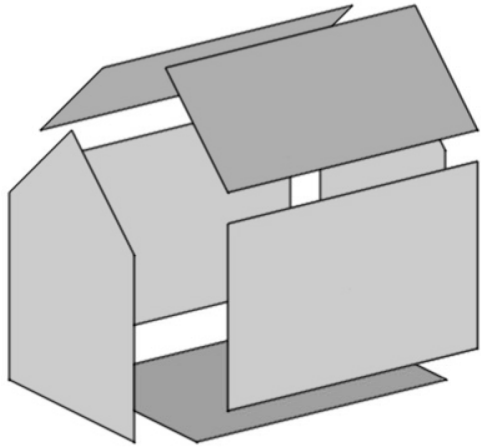
TU Delft

Entity-relationship model

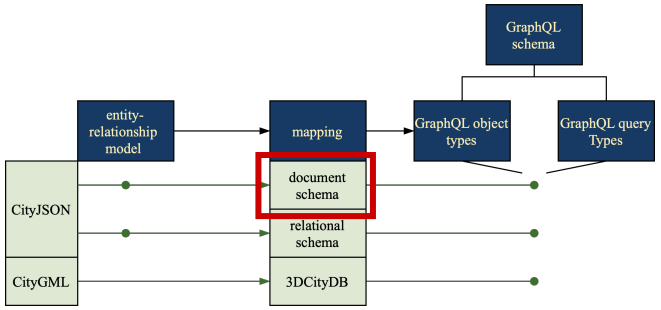


- Dependent/ independent entities
- Many-to-many relationship
- Geometry challenge
- Semantics challenge
- JSON





Document schema

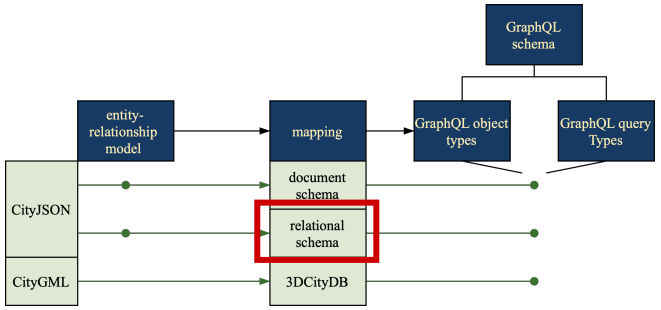


- Dependent/ independent entities
- Many-to-many relationship
- Geometry challenge
- Semantics challenge

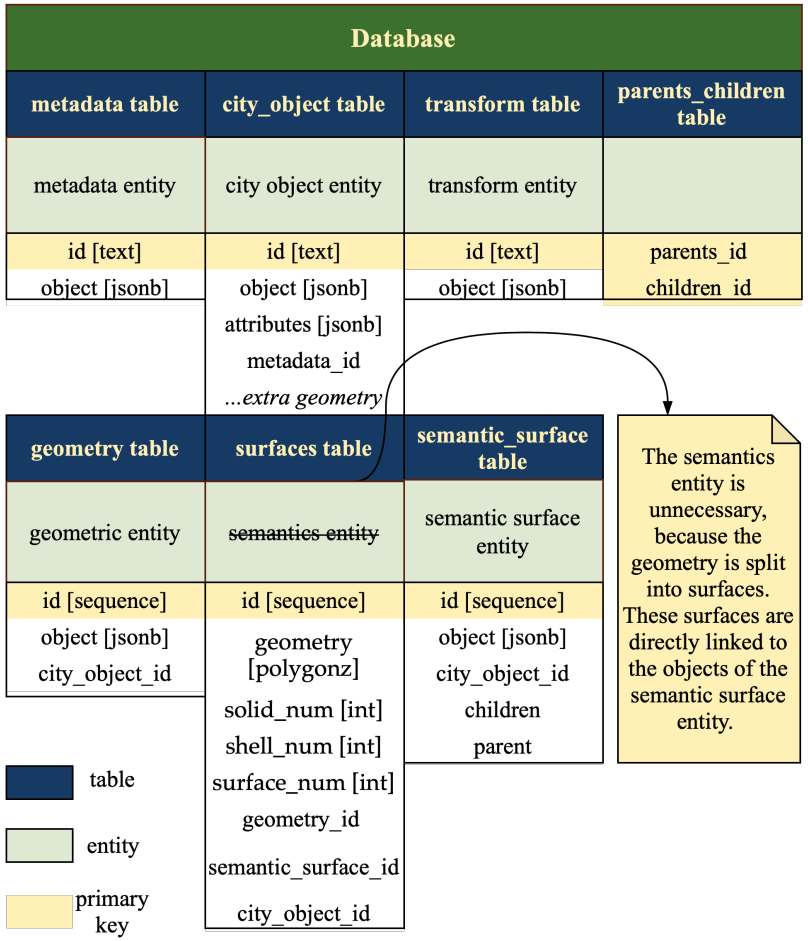
Database		
Collection	**Collection**	**Collection**
metadata entity	city object entity	transform entity

nested document	**nested document**	**nested document**	**nested document**
attributes entity	geometric entity	semantics entity	semantic surface entity

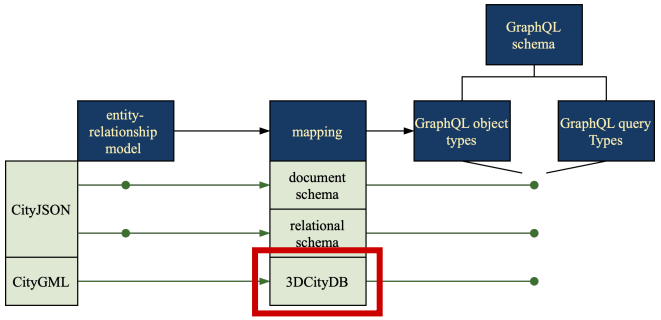
Relational schema



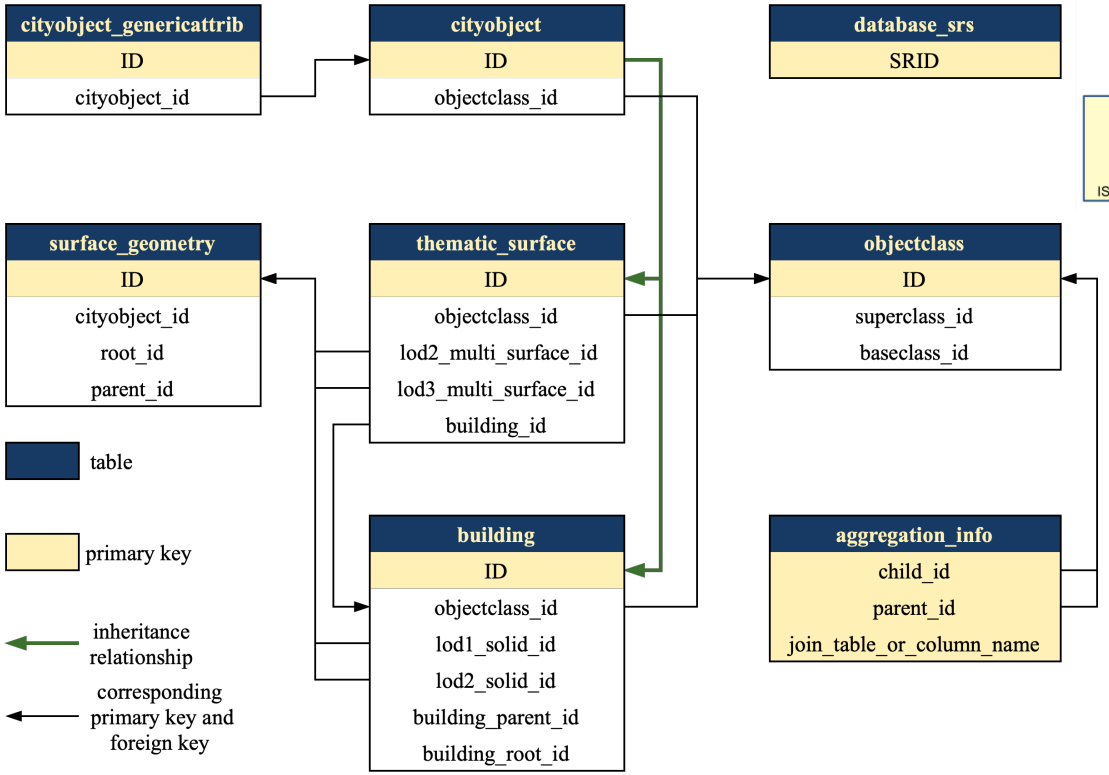
- Dependent/ independent entities
- Many-to-many relationship
- Geometry challenge
- Semantics challenge

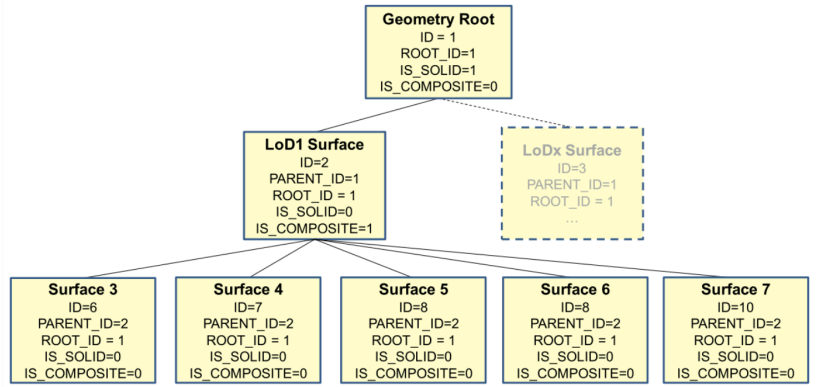


3DCityDB



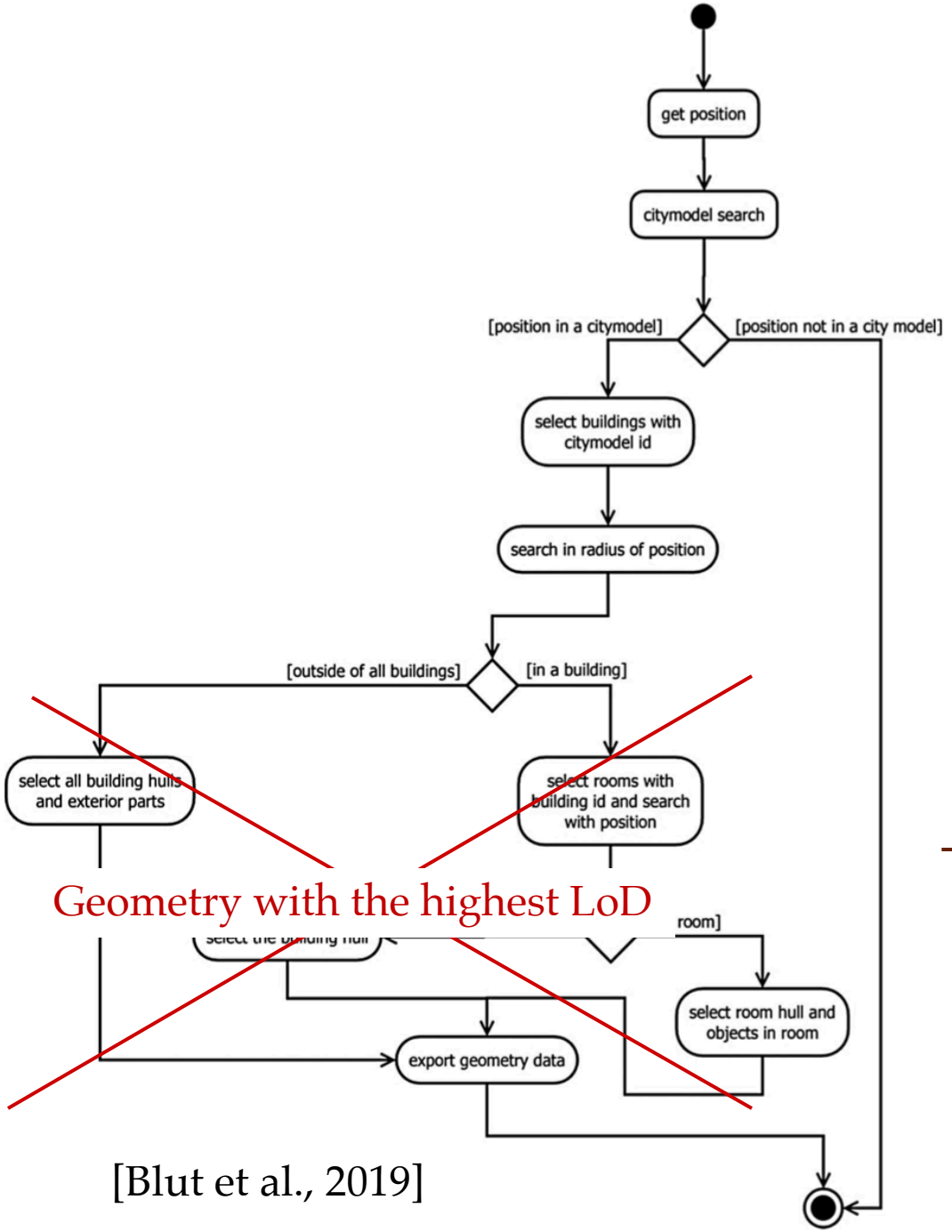
- Inheritance relationship
- Generic attributes
- Geometries





[Kolbe, 2019]

Selection process



→ 7 queries

[Blut et al., 2019]

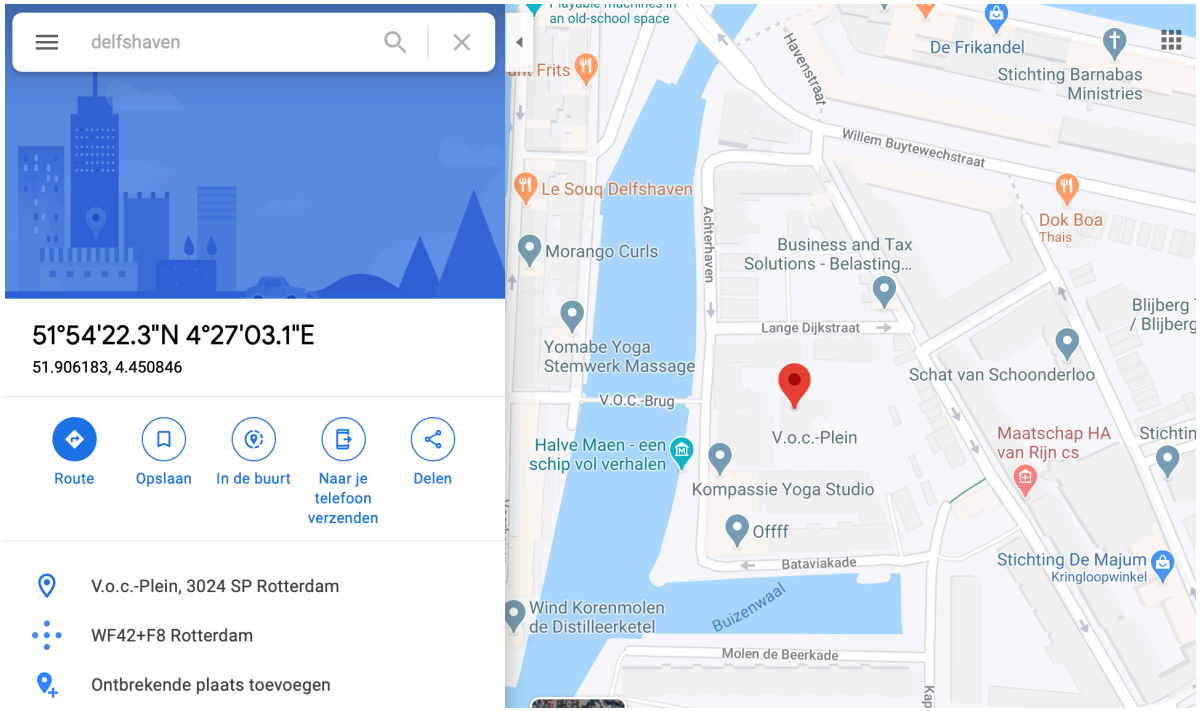
The 7 queries

query number and name	argument	operations	filter	returned data
1: location	position with the fields lat, long and alt			location with the fields latitude, longitude and altitude
2: citymodel	position with the fields lat, long and alt	intersection		city model ID
3: radius100	position with the fields lat, long and alt	within 100 meter	city object type = Building	city object IDs
4: inside	position with the fields lat, long and alt	intersection	city object type = Building	city object ID
5: maxlod	city object ID			ID of the geometry with the highest LoD
6a: cityobjects	city object ID		ID-based	return the ID of the city object
6b: cityobjects	city object ID		ID-based	return the ID and attributes of the city object
6c: cityobjects	city object ID		ID-based	return the ID and geometries of the city object
7: surfaces	surface ID or other		ID-based or other	return the surface with the related semantic surface object

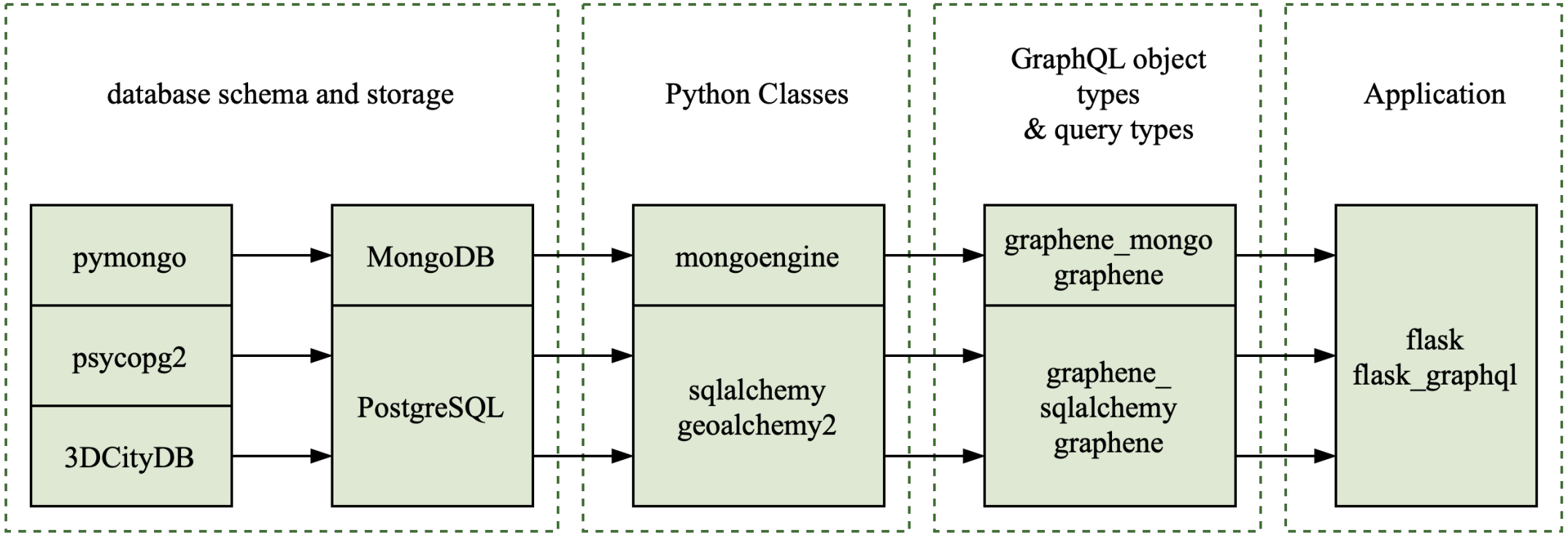
TU Delft

Data

Denhaag	2.6 MB	EPSG:4715	parent-child relationship	
Delfshaven	1.4 MB	EPSG: 28992		
Potsdam	15.7 MB	EPSG: 25833	multiple LoDs	presentLoDs



Software

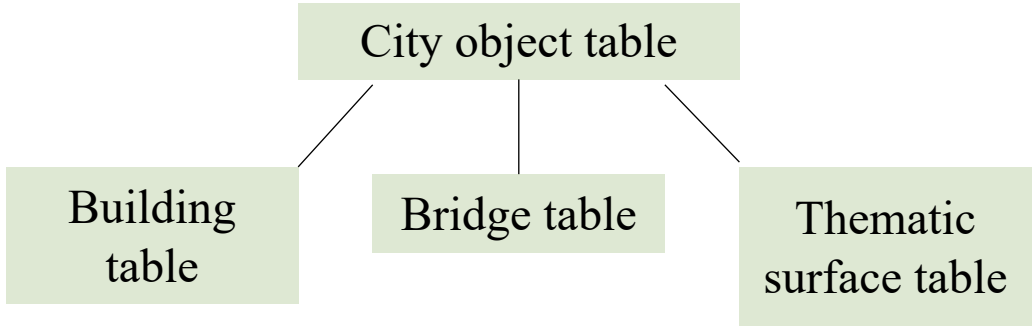

database schema and storage
Python Classes
GraphQL object types & query types
Application
pymongo
psycopg2
3DCityDB
MongoDB
PostgreSQL
mongoengine
sqlalchemy geoalchemy2
graphene_mongo graphene
graphene_ sqlalchemy graphene
flask flask_graphql

CityJSON in PostgreSQL

	id [PK] text	object jsonb	attributes jsonb
1	UUID_93f...	{"type": "Building...	{"Geomtype": 1, "creation...
2	UUID_c98...	{"type": "Building",...	{"class": "BB01", "Region"...
3	UUID_a4a...	{"type": "Building...	{"Geomtype": 1, "creation...
4	UUID_ba0...	{"type": "Building",...	{"class": "BB07", "Region"...

```
{ cityobjects {
    id
}}
```

CityGML in PostgreSQL with 3DCityDB




```
{ cityobjects {
    building {
        id
}}}
```

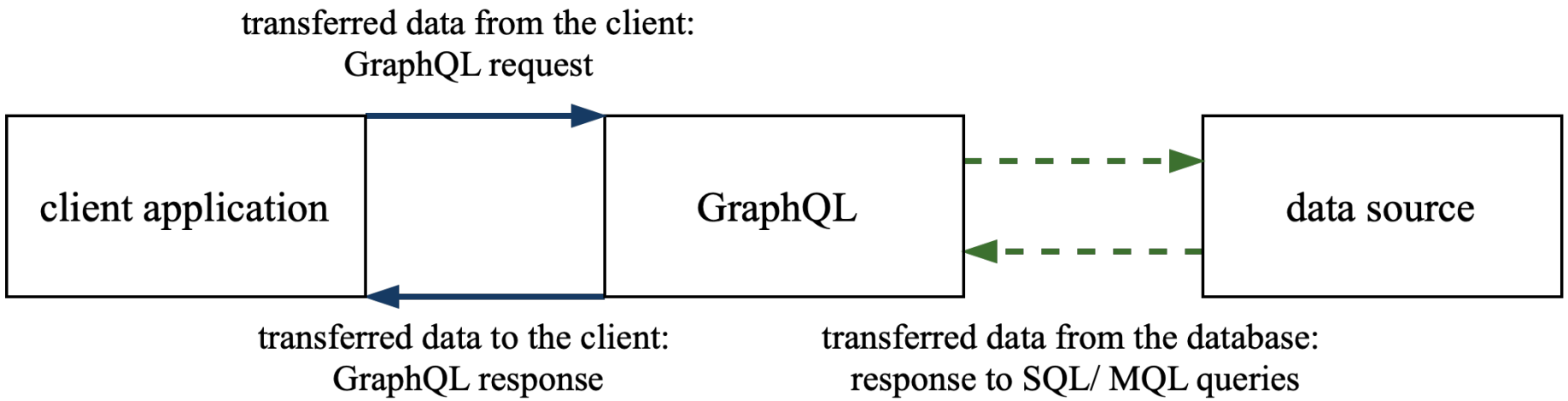
CityJSON in PostgreSQL	CityJSON in MongoDB
	"presentLoDs":{"2.0":145865}
one city object – multiple tables	one city object – one document
reference transformations	no reference transformations
Maximum field size = 1GB	Maximum document size = 16MB

```
MultiSurface:
boundaries = ListField(ListField(ListField(IntField())))
MultiSolid:
boundaries = ListField(ListField(ListField(ListField(ListField(IntField())))))
```

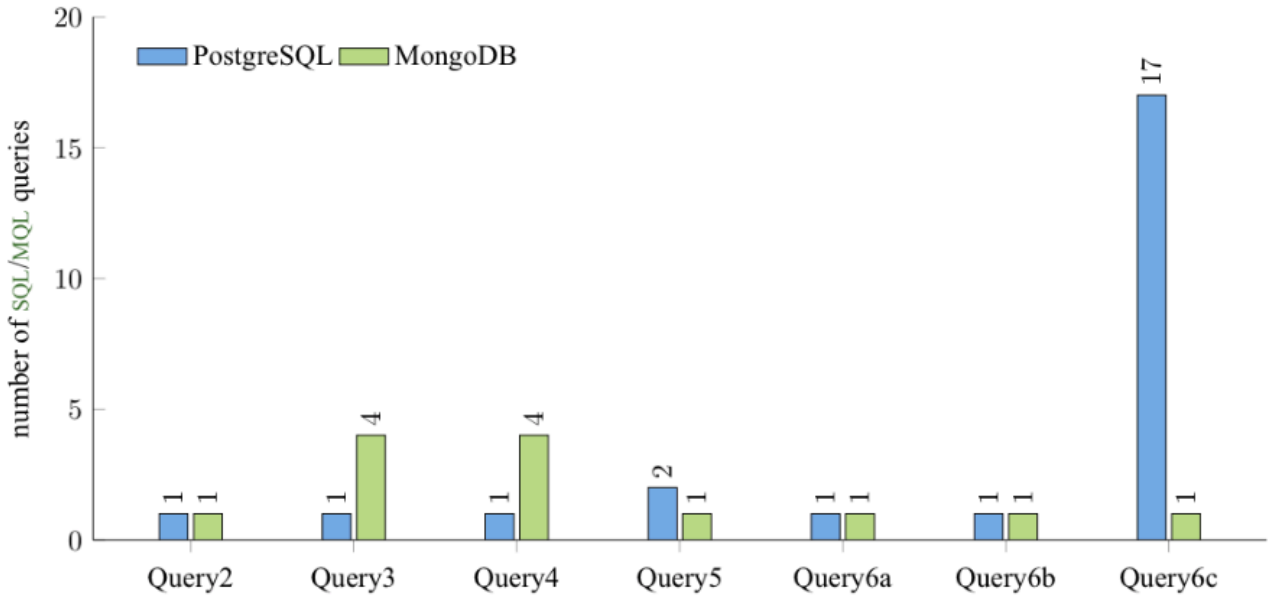
TU Delft

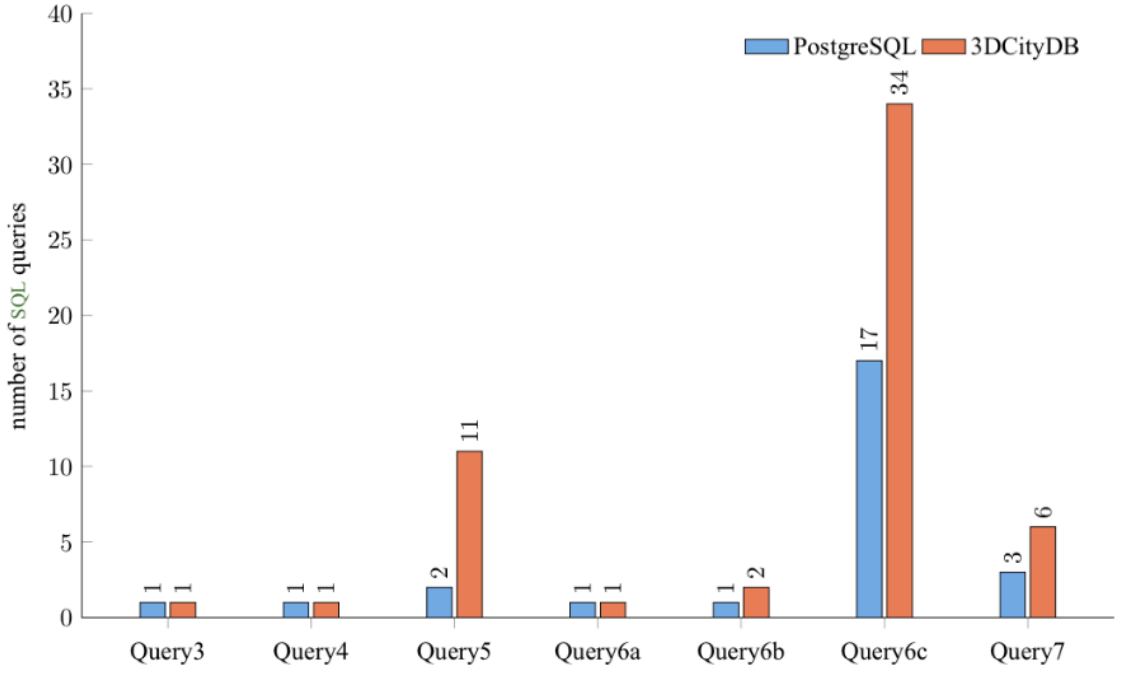
Experiments

- Retrieval times
- Request and response sizes
- Over-fetching



Results


PostgreSQL
MongoDB
number of SQL/MQL queries
0
5
10
15
20
Query2
Query3
Query4
Query5
Query6a
Query6b
Query6c
1
1
1
4
1
4
2
1
1
1
1
1
17
1


PostgreSQL
3DCityDB
number of SQL queries
0
5
10
15
20
25
30
35
40
Query3
Query4
Query5
Query6a
Query6b
Query6c
Query7
1
1
1
1
2
11
1
1
1
2
17
34
3
6

Results

- More queries lead to higher retrieval times
  - Possible due to ORM/ODM → not using joins
- More overfetching for MongoDB and 3DCityDB than PostgreSQL
  - Possible due to ORM/ODM → not selecting fields

How suitable are MongoDB and PostgreSQL for the storage and querying of CityJSON using GraphQL?

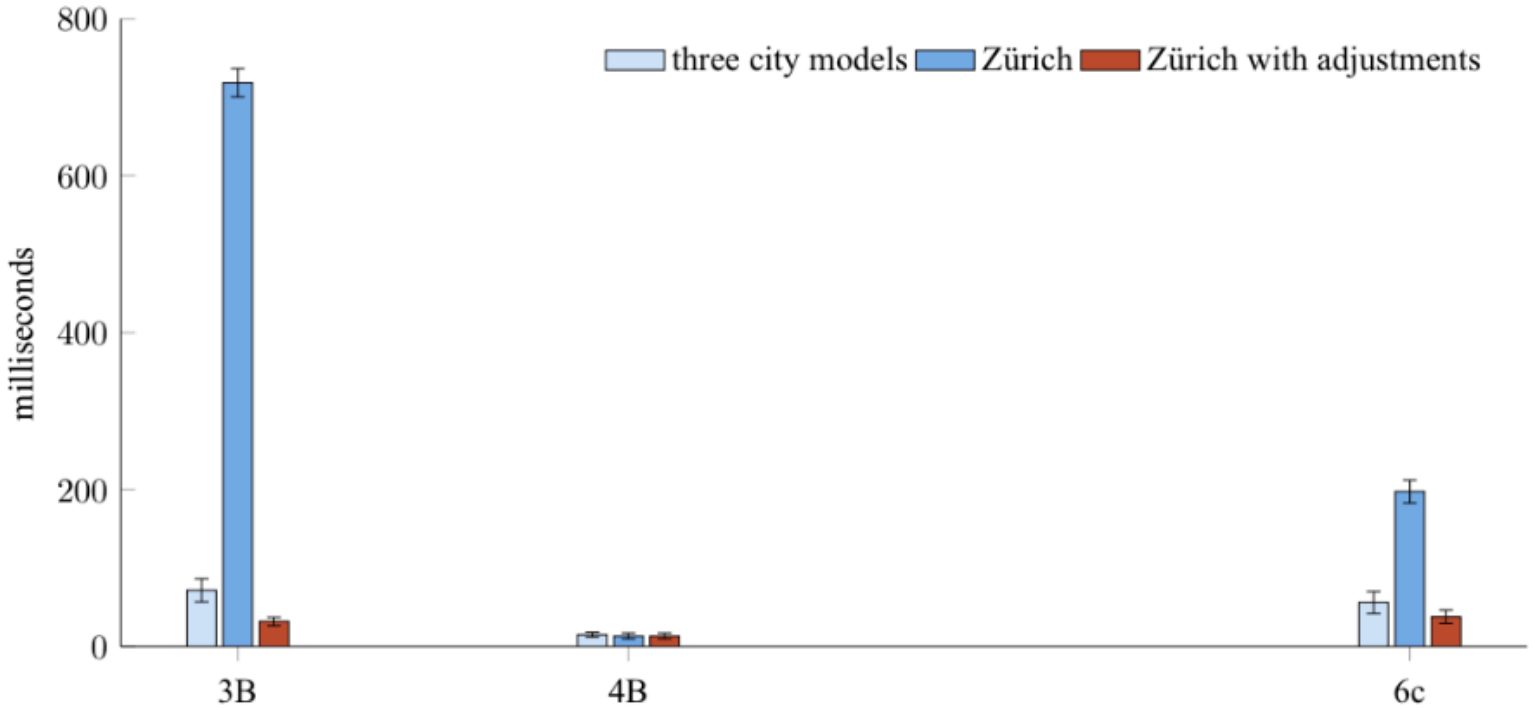
- No real signs yet that MongoDB and PostgreSQL are not suitable
  - Vertices list
  - Attribute presentLoDs
  - reference transformations
- Integration of JSON
  - Attributes are mapped more flexibly → less tables → less queries/joins → lower retrieval times
  - Support for fields with varying data types
    - + support to map the hierarchy of arrays
    - - GraphQL is less flexible
  - Querying a JSON attribute -→ higher retrieval times = not investigated

Recommendations

- Implement the SQL/MQL queries as efficient as possible without being connected to GraphQL first
  - + better understanding database performance
  - + better understanding abilities GraphQL
- A more general understanding of the suitability for all use cases could be provided with a framework that tests more types of queries.
  - Aggregate queries
  - Queries on GeoJSON objects in MongoDB
  - Queries on JSON attributes in PostgreSQL and MongoDB

Recommendations

	Three datasets together	Zürich
Size (MB)	19.7	292.8
number of objects	6258	198699



Questions?

Spatial queries per implementation

Use one city model instead of multiple

Store convex hull as GeoJSON object

PostgreSQL			MongoDB
supports many reference systems geometrical spatial operations on 2D and 3D geometries geographical spatial operations on 2D geometries			supports GeoJSON objects in 2D with WGS84 as global reference system
CityGML in PostgreSQL with 3DCityDB	**CityJSON in PostgreSQL**		**CityJSON in MongoDB**
envelope transformed in the query (in the database)	convex hull transformed in the query (in the database)	global convex hull transformed before query	hierarchy of arrays transformed in query (outside the database)
index on envelope	additional indexes: index on convex hull index on global convex hull		