THESIS RESULTS PHASE 1

LUCA CASSENTI

TABLE OF CONTENT

Summary

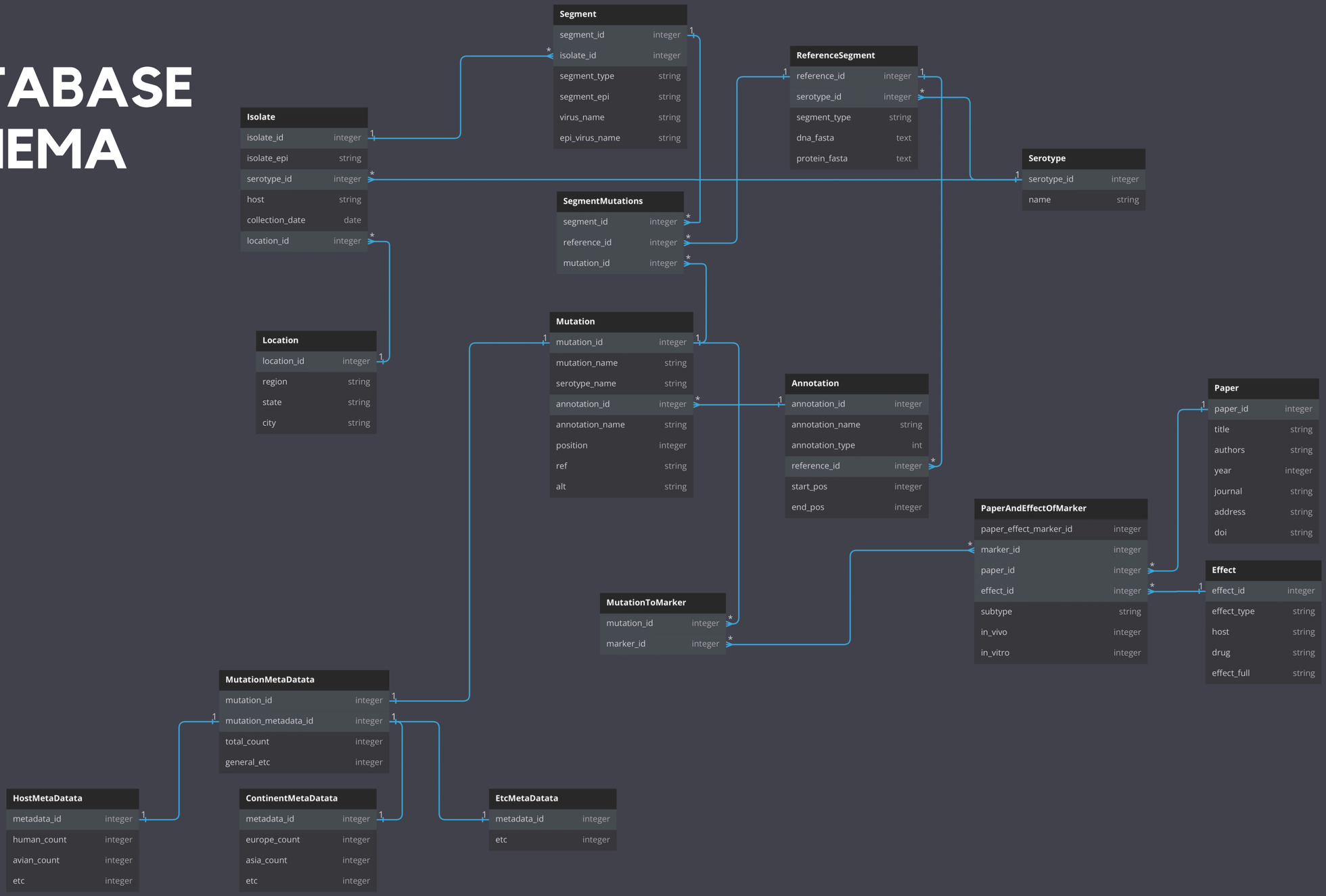
This first phase of the thesis focused on the collection, analysis, and refinement of data concerning mutations within the avian flu virus. At first only for the H5N1 strain, but then involving every strain available.

This phase was mostly about data cleaning and preliminary investigations to establish a robust foundation for the creation of the final tool with the objective to decipher the evolution of the virus, with focus on factors such as transmission dynamics, virulence traits, and possible implications for vaccine development.

Work started with an analysis of requirements given by the IZSVe, with which we collaborate, to clearly define the scope of the project. Then proceed with the necessary analysis of the data, the creation of a database schema and the aggregation of data with some example queries.

TABLE OF CONTENT

DATABASE SCHEMA

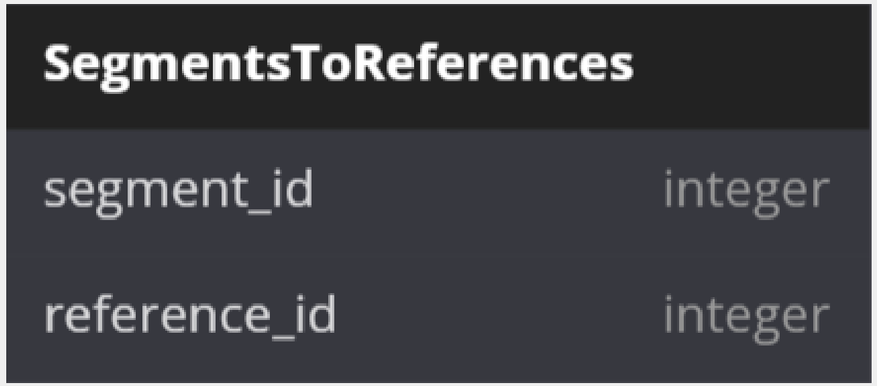


MAIN FEATURES

- COMPLETE GRASP OF THE FLU DOMAIN
- SEARCH ON MUTATIONS, GROUPS OF, AND ANY MAIN CONCEPT
- QUERY WITH STATISTICAL RELEVANCE
- USE OF SECONDARY STRUCTURES TO HASTEN QUERIES
- COMPLETE INTEGRATION OF IZSVE INITIAL WORK

EXCLUDED BRIDGE TABLE

WHEN DESIGNING THE DATABASE SCHEMA, THE FOLLOWING BRIDGE TABLE FOR SEGMENTS TO REFERENCES WAS NOT INCLUDED AS FROM A PRACTICAL STANDPOINT, IT WOULD HAVE UNNECESSARILY COMPLICATED THE QUERIES.



DESPITE THAT IT IS IMPORTANT TO CONSIDER IT LOGICALLY AS TO BETTER REPRESENT THE DOMAIN.

IN FACT, ALTHOUGH IT WOULD BE POSSIBLE TO RECONSTRUCT THE TABLE FROM A JOIN CLAUSE, THE LOGICAL BOND BETWEEN THE 'SEGMENT' AND THE 'REFERENCE SEGMENTS' ENTITIES HAS A BIG SIGNIFICANCE IN THE RESEARCH OF MUTATIONS.

TABLE OF CONTENT

Three Types of Data

annotation.xlsx

0-17 of 17

	C0	C1
1	annotation_id	annotation_name
2	0	PB2
3	1	PB1
4	2	PB1-F2
5	3	PA
6	4	PA-X
7	5	PA-X
8	6	HA1
9	7	HA2
10	8	NP

```
>EPI1530142|PB2|A/ruddy_turnstone/Del
caaatatattcaatatggagagaataaaagaattaag
accactgttgaccacatggccataattaaaaagtaca
gatgatggcaatgaaatatccaatcacagcagacaag
tcctctggagcaaaacaaacgatgccgggtcagaccg
ggaccaacaacaagtacagttcactatccaaaggtat
ctttggccctgtacacttcagaaatcaagttaagata
ccaaagaggcgcaggatgtaatcatggaagttgtctt
ttgacaataacaaaggagaagaaggaagaactccagg
agagctggtccgaaagacaaggtttctcccagtggct
aagggacatgctgggagcagatgtacactccgggagg
gccaggaacatagtaagaagggcaacagtgtcagcag
tggaggaataagaatggtagacattcttcggcaaaat
gcttgaggattagctcatccttcagctttggtgggtt
gaagtgcttacgggcaacctccaaacattgaaaataa
```

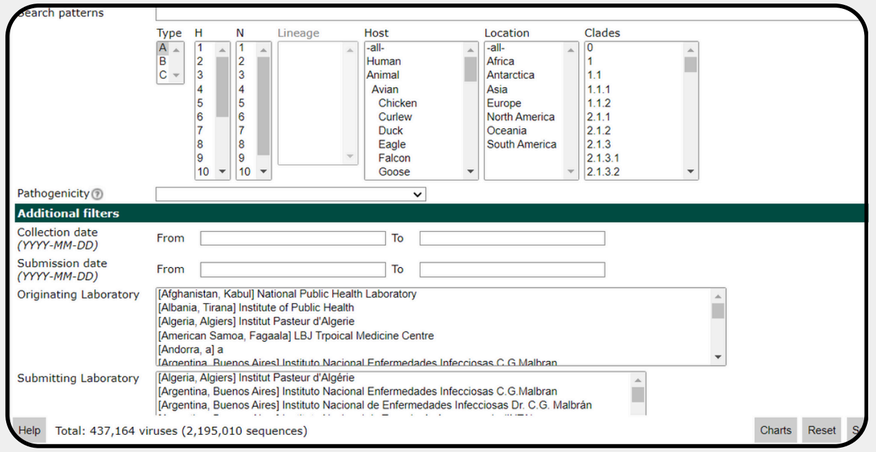
01 - STATIC VIRUS DATA

02 - MUTATION DATA

03 - PAPERS DATA

DATA FROM GISAID

We downloaded the flu data from GISAID, a global database that provides detailed genomic data. GISAID categorizes flu data by types (A, B, C), with our focus exclusively on type A. The database further distinguishes data by the hemagglutinin (H) and neuraminidase (N) segments, among other filters. For our analysis, we specifically filtered the data based on the H and N segment numbers, such as H5N1, ensuring our dataset was precise and relevant to our study objectives.

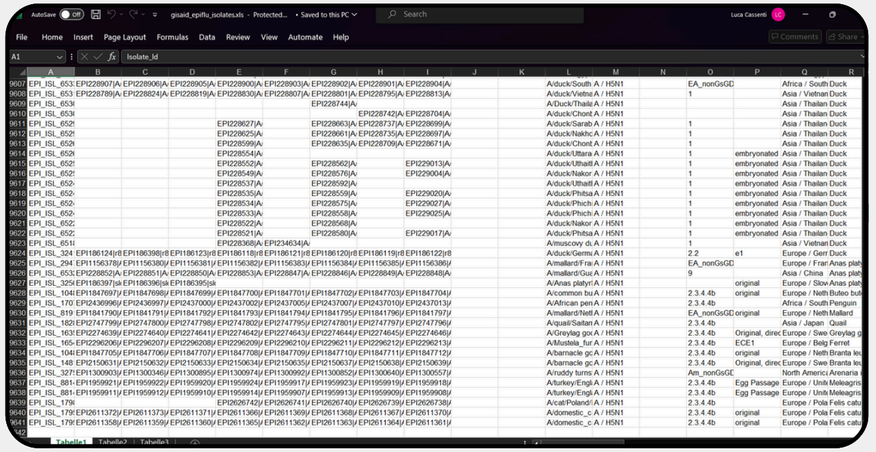


In particular the GISAID database has a total of ~450K Isolates for the type A, for a total of ~2.2 million Segment sequences, which roughly translates to 5 Segments per Isolates, despite each Isolate could have as much as 8 Segments (HA, NA, PB2, PB1, PA, NP, NS and MP).

The most common hemagglutinin and neuraminidase types are the H1N1, which accounts for ~150K Isolates, and the H3N2, with ~190K Isolates. Third place goes to H5N1 with ~20K Isolates, well beyond the other two.

DATA FROM GISAID

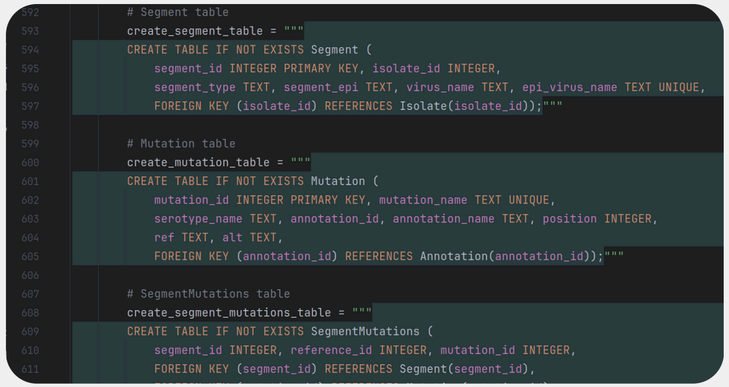
Each Isolate can be downloaded in different ways to extract different information:



- It is possible to either extract its metadata, which includes data such as the available segments for that isolate, it's subtype (ex: H1H5), where it was retrieved, from which host, etc. The totality of the metadata is composed by qualitative data, thus a quantitative analysis cannot be performed
- Otherwise it is possible to extract either the DNA (nucleotidic) or Protein sequences. This data is the exact sequence that was extracted from the isolate and is divided by the different segments. It is this kind of data that is then used to find mutations in the given Isolate by comparing the sequences with their respective references, divided by segment type and subtype.

DB CREATION

1. DB SCHEMA



```
# Segment table
create_segment_table = """
CREATE TABLE IF NOT EXISTS Segment (
    segment_id INTEGER PRIMARY KEY, isolate_id INTEGER,
    segment_type TEXT, segment_epi TEXT, virus_name TEXT, epi_virus_name TEXT UNIQUE,
    FOREIGN KEY (isolate_id) REFERENCES Isolate(isolate_id));"""

# Mutation table
create_mutation_table = """
CREATE TABLE IF NOT EXISTS Mutation (
    mutation_id INTEGER PRIMARY KEY, mutation_name TEXT UNIQUE,
    serotype_name TEXT, annotation_id, annotation_name TEXT, position INTEGER,
    ref TEXT, alt TEXT,
    FOREIGN KEY (annotation_id) REFERENCES Annotation(annotation_id));"""

# SegmentMutations table
create_segment_mutations_table = """
CREATE TABLE IF NOT EXISTS SegmentMutations (
    segment_id INTEGER, reference_id INTEGER, mutation_id INTEGER,
    FOREIGN KEY (segment_id) REFERENCES Segment(segment_id),
```

The first step in the creation of the database consists in translating the DB Schema created for the domain into a SQL schema using CREATE TABLE commands and by correctly linking the various tables using foreign keys.

2. GISAID DATA

Secondly, we need to download all the necessary data from the GISAID database. In particular we needed to download as many Isolates from type A, for a total of ~450K Isolates.

For each Isolate we downloaded the DNA Sequence and its metadata, in order to fill up the database with the necessary information both on the Isolate and its Segments, by using the metadata, and on the Mutations, by using the DNA sequences, by aligning them.

All the data was downloaded manually, as the subdivision by subtype on GISAID was only a slight problem for the H1N1 subtype and the H2N3 subtype.

3. REFERENCES

The third step was to obtain the references sequences necessary to obtain the mutation from each sample sequence.

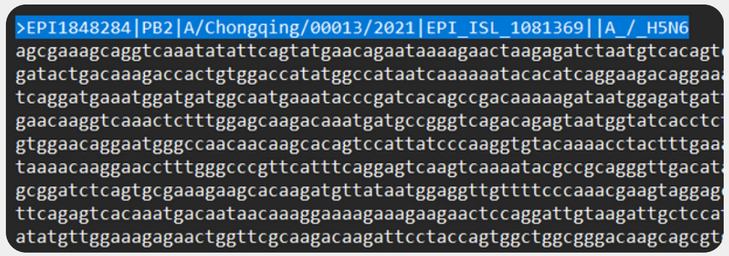
These sequences are what could be considered the ancestors of all the analyzed data, from which all the flu samples have evolved and mutated.

Each sample can have multiple references in theory but usually only one is necessary. Furthermore these references are categorized by subtype and segment type.

In particular, for the H5N1 subtype we used the references provided by the IZSVe

4. METADATA INTEGRATION

Before processing the mutations, each sequence is correlated with its metadata, in order to ensure consistency in the database by classifying each sample once and into the proper categories for future queries.



166	EPI_ISL_2081527	EPI1916407|A/red_fox/England/AVP-M1-21-(	EPI191(
167	EPI_ISL_4063607	EPI1973694|A/mink/China/F0130m/2018	EPI1973
168	EPI_ISL_16466440	EPI2287045|A/Changsha/1/2022	EPI2287
169	EPI_ISL_1081370	EPI1848292|A/Anhui/2021-00011/2020	EPI1848
170	EPI_ISL_1081369	EPI1848284|A/Chongqing/00013/2021	EPI1848
171	EPI_ISL_180669		
172	EPI_ISL_19055420	EPI3210827|A/large-billed crow/Iwate/0303G(	EPI3210
173	EPI_ISL_19055419	EPI3210819|A/large-billed crow/Iwate/0302G(	EPI3210

5. VARIANT CALLING

The final step in the creation of the database data consists in finding the mutations of each sample using variant calling.

Variant calling is a crucial process in genomic analysis, used to identify genetic differences between sequences to detect mutations. The workflow begins with Augur, a toolkit that aligns the sequences in their DNA form, creating a standardized reference framework. Once aligned, the coding sequences for each protein (ex: M1 and M2 for segment MP) are extracted from these DNA sequences. These coding sequences are then translated into their corresponding amino acid sequences, reflecting the actual proteins produced by the organism.

The next step involves further aligning these amino acid sequences. This detailed alignment allows for the precise identification of mutations by highlighting differences from a reference sequence.

By focusing on the amino acid level, we can better understand the functional impacts of these mutations, as changes in protein structure correspond to biological consequences.

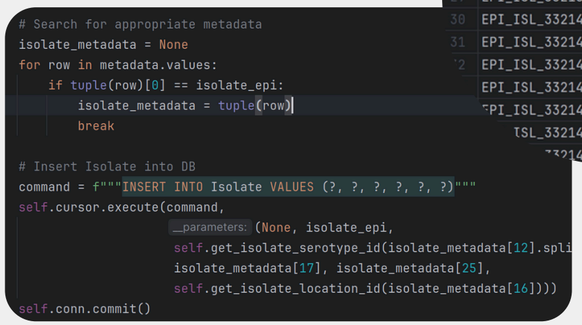
DB FILL

While the sample data and associated metadata are utilized by the software to create entries for the mutations, the system simultaneously checks and creates entries for the Isolate and Segment tables if they do not already exist.

These tables contain comprehensive information about each sample and its respective genomic segments.

Additionally, the data correlating each sample segment with its identified mutations and the reference sequence used is systematically organized into a bridge table. This bridge table effectively links all relevant information, allowing for efficient tracking and analysis of the relationship between samples, segments, and their mutations.

ISOLATES




```
# Search for appropriate metadata
isolate_metadata = None
for row in metadata.values:
    if tuple(row)[0] == isolate_epi:
        isolate_metadata = tuple(row)
        break

# Insert Isolate into DB
command = f"""INSERT INTO Isolate VALUES (?, ?, ?, ?, ?, ?)"""
self.cursor.execute(command,
                    parameters: (None, isolate_epi,
                    self.get_isolate_serotype_id(isolate_metadata[12].spli
                    isolate_metadata[17], isolate_metadata[25],
                    self.get_isolate_location_id(isolate_metadata[16])))
self.conn.commit()
```

+ CROSS REFERENCING WITH METADATA

SEGMENTS

```
# Check if segment exists
try:
    self.cursor.execute( sql: "SELECT * FROM Segment WHERE epi_virus_name = ?",
                         parameters: (f"{header_dict['segment_id']}",))
    segment = self.cursor.fetchone()
    if segment:
        return segment["segment_id"]  # return id for SegmentMutations
except sqlite3.OperationalError:
    pass

# Insert Segment into DB
command = f"""INSERT INTO Segment VALUES (?, ?, ?, ?, ?, ?)"""
self.cursor.execute(command,
                    parameters: (None,
                    header_dict["isolate_id"], header_dict["segment_type"],
                    header_dict["segment_epi"], header_dict["virus_name"], header_dict["segment_id"]
# Return id for SegmentMutations
self.conn.commit()
return self.cursor.lastrowid
```

MUTATIONS

```
def insert_mutation(self, serotype, annotation_id, mutation):

    # Find gene name by id
    try:
        self.cursor.execute( sql: "SELECT * FROM Annotation WHERE annotation_id = ?",  parameters:
        annotation = self.cursor.fetchone()
        annotation_name = annotation["annotation_name"]
    except sqlite3.OperationalError:
        annotation_name = None

    position = int(mutation[:mutation.find('_')])
    ref = mutation[mutation.find('|') - 1:mutation.find('|')]
    alt = mutation[mutation.find('|') + 1:]
    mutation_name = f"{serotype}:{annotation_name}:{ref}{position}{alt}"

    # Insert Segment into DB
    command = f"""INSERT INTO Mutation VALUES (?, ?, ?, ?, ?, ?, ?)"""
    self.cursor.execute(command,
                        parameters: (None, mutation_name, serotype, annotation_id,
                        annotation_name, position, ref, alt))
```

+ BRIDGE TABLE MUTATIONS & SEGMENTS

NOTE: VIEW CREATION FOR DATA TRANSFER

```
SELECT
row_number() OVER (ORDER BY pwi.n)
paper_effect_marker_id,
me.marker_id, pwi.n as paper_id,
ewi.n as effect_id,
me.subtype, me.in_vivo, me.in_vitro
FROM markers_effects me JOIN
papers_with_ids pwi ON me.paper_id = pwi.id JOIN
effects_with_ids ewi ON me.effect_name = ewi.effect_full
```

	paper_effect_marker_id	marker_id	paper_id	effect_id	subtype
1	1	149	4	24	H5N1
2	2	149	4	80	H5N1
3	3	149	4	82	H5N1
4	4	134	12	65	H5N1
5	5	34	13	34	H5N1
6	6	34	13	66	H5N1
7	7	34	13	74	H5N1
8	8	107	14	14	H5N1
9	9	107	14	18	H5N1
10	10	107	14	22	H5N1
11	11	131	14	62	H5N1
12	12	131	14	77	H5N1
13	13	71	15	62	H9N2
14	14	71	15	77	H9N2
15	15	75	16	73	H1N2

In our database migration strategy, I've prioritized data consistency between IZSVe's database and our new system. To achieve this, I've developed the following views:

1. PEM View: This view combines paper, effect, and marker data, ensuring seamless integration.

2. Markers Summary View: Summarizing markers and their associated mutations, this view provides a comprehensive overview.

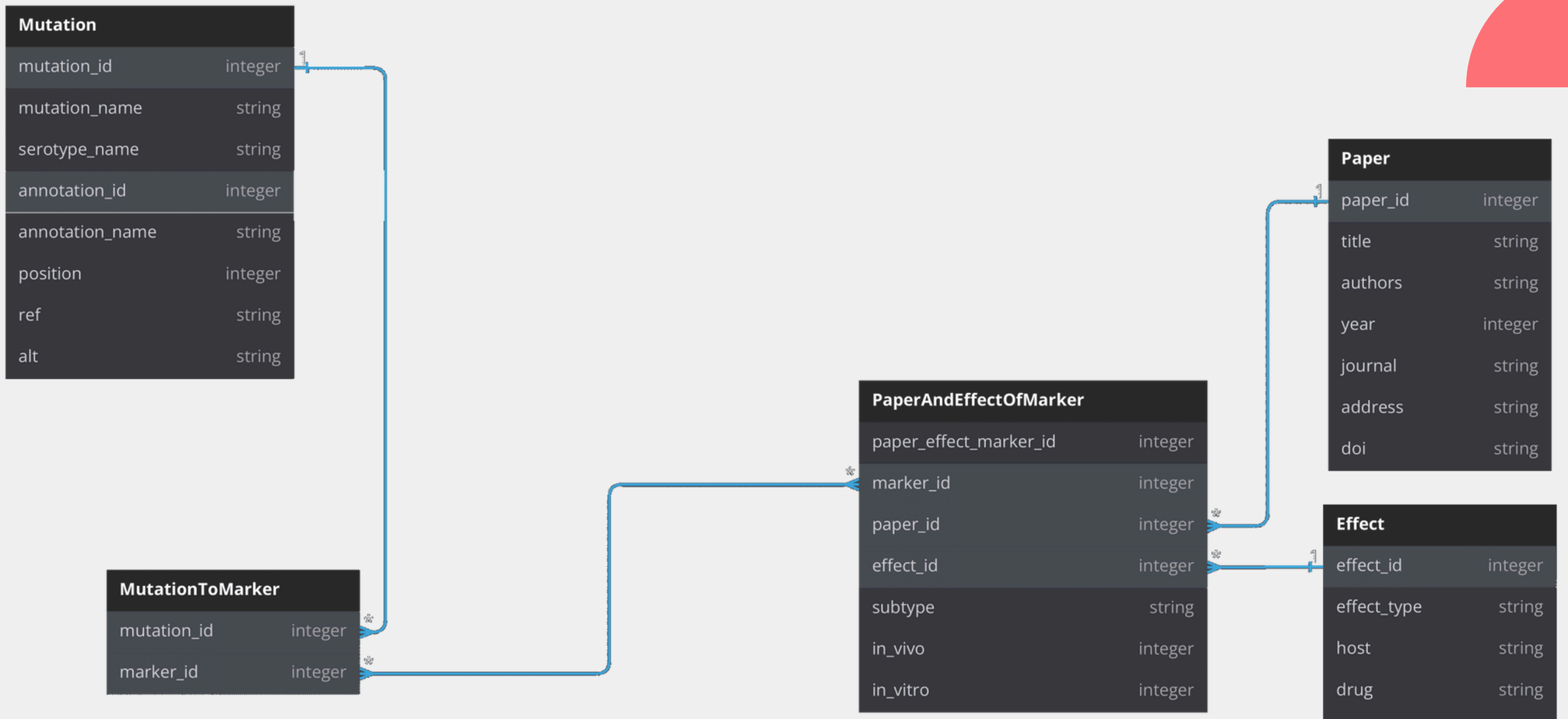
3. Papers with IDs View: By assigning unique numerical identifiers to papers and filtering out those without a DOI, this view establishes consistency in paper indexing.

4. Effects with IDs View: Assigning unique identifiers to effects streamlines data organization.

These views collectively enhance data integrity, simplifying querying and analysis while bolstering overall database management efficiency.

TABLE OF CONTENT

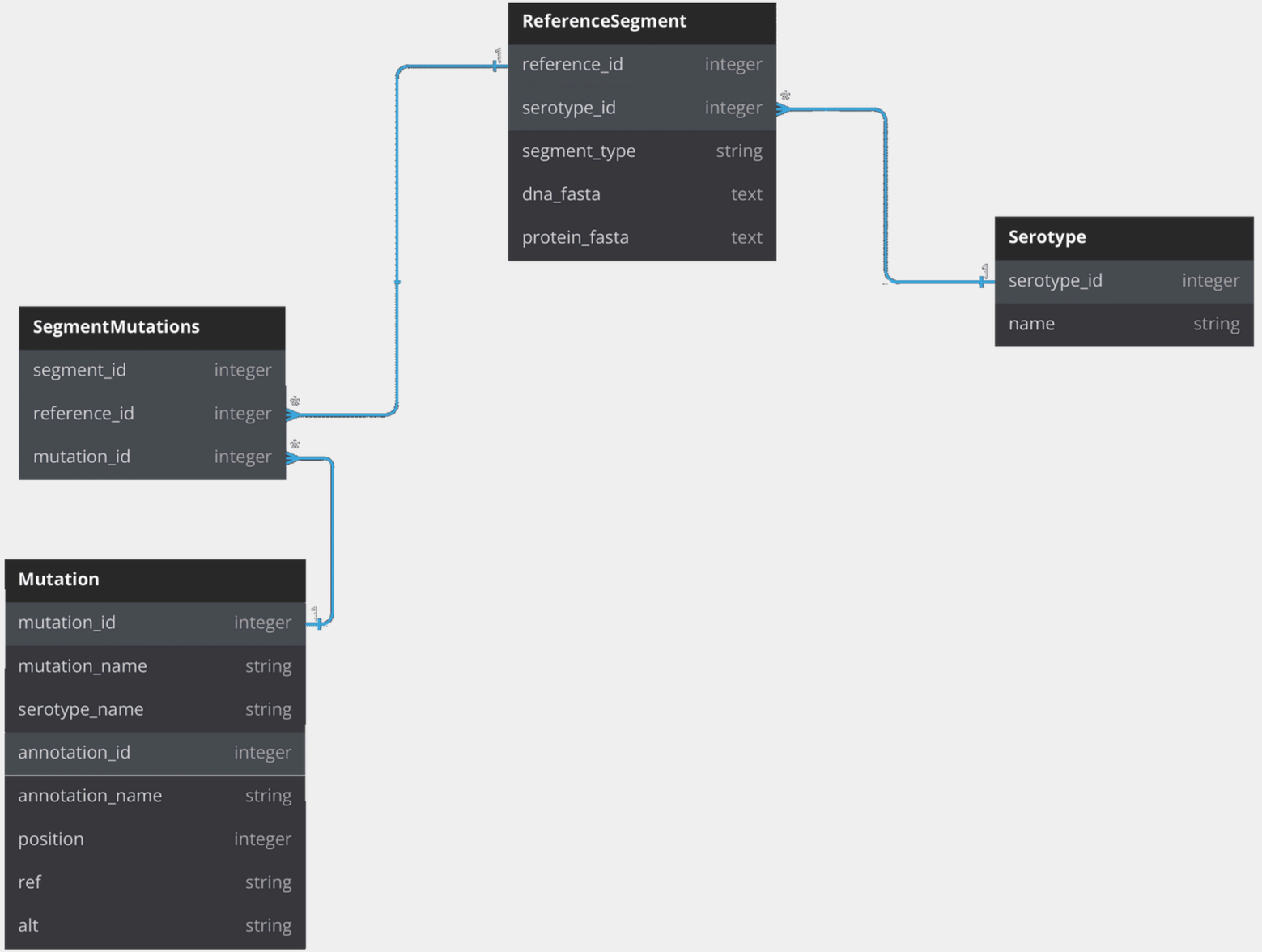
QUERY 1: GET ALL EFFECTS ASSOCIATED WITH A PARTICULAR MUTATION



QUERY 1: GET ALL EFFECTS ASSOCIATED WITH A PARTICULAR MUTATION

```
SELECT * FROM Effect e
WHERE e.effect_id in (
    SELECT pem.effect_id
    FROM MutationsToMarker mtm
    JOIN Marker m ON mtm.marker_id = m.marker_id
    JOIN PaperAndEffectOfMarker pem ON m.marker_id = pem.marker_id
    WHERE mtm.mutation_id = 6501)
```

QUERY 2: GET ALL MUTATIONS ASSOCIATED WITH ONE SEROTYPE



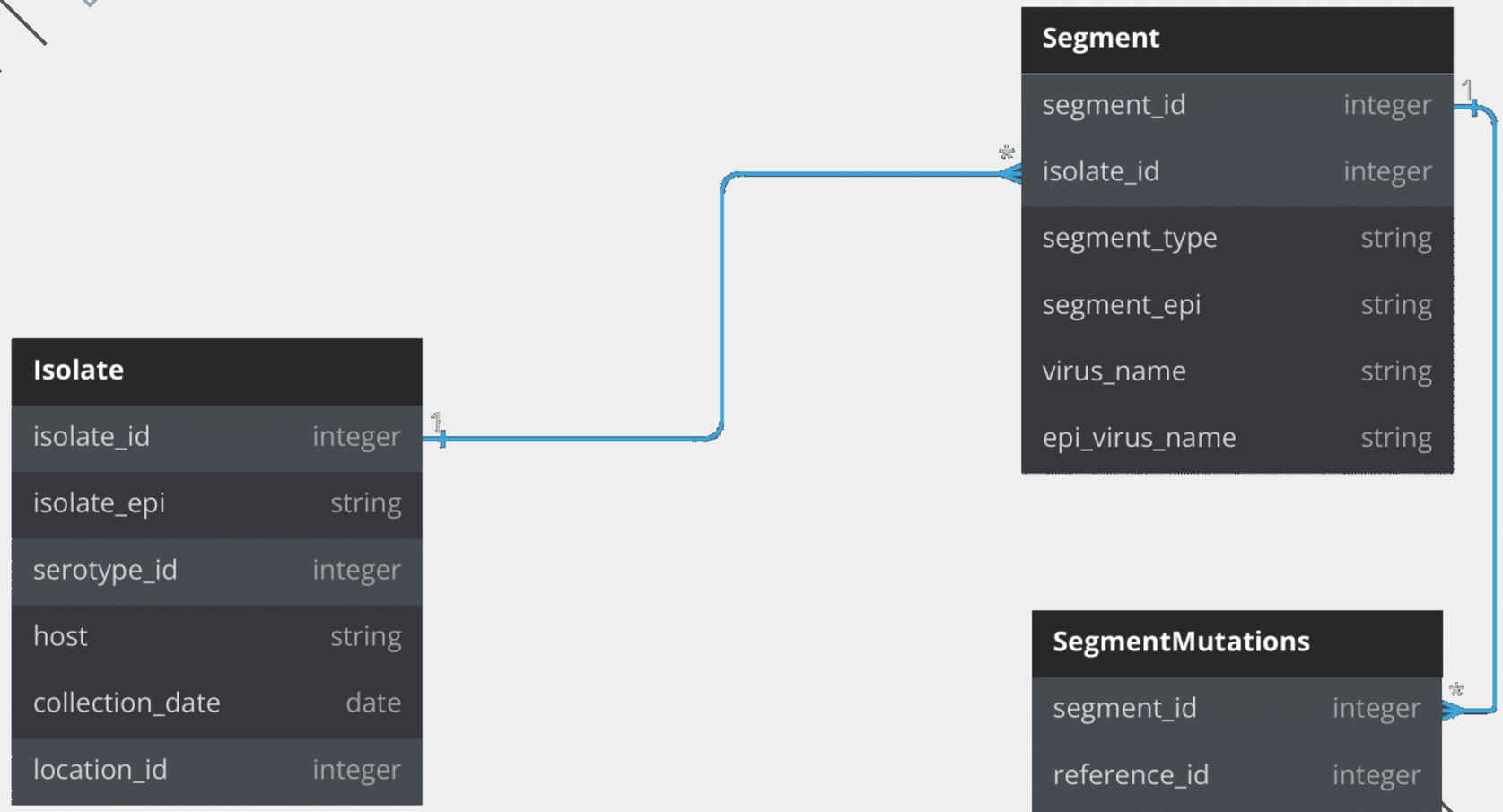
QUERY 2: GET ALL MUTATIONS ASSOCIATED WITH ONE SEROTYPE

```
SELECT DISTINCT mut.mutation_name, sm.mutation_id
FROM SegmentMutations sm
JOIN ReferenceSegment rs on sm.reference_id = rs.reference_id
JOIN Serotype sero ON sero.serotype_id = rs.serotype_id
JOIN Mutation mut ON mut.mutation_id = sm.mutation_id
WHERE sero.name = "H5N1"
```

```sql
SELECT DISTINCT mut.mutation_name, sm.mutation_id, mut.mutation_name
FROM SegmentMutations sm
JOIN ReferenceSegment rs on sm.reference_id = rs.reference_id
JOIN Serotype sero ON sero.serotype_id = rs.serotype_id
JOIN Mutation mut ON mut.mutation_id = sm.mutation_id
WHERE sero.name = "H5N1"
```

	mutation_name	mutation_id	mutation_name
1	H5N1:HA1:K3R	1	H5N1:HA1:K3R
2	H5N1:HA1:K5R	2	H5N1:HA1:K5R
3	H5N1:HA1:E7N	3	H5N1:HA1:E7N
4	H5N1:HA1:R8K	4	H5N1:HA1:R8K
5	H5N1:HA1:N8S	5	H5N1:HA1:N8S
6	H5N1:HA1:G8D	6	H5N1:HA1:G8D
7	H5N1:HA1:S9N	7	H5N1:HA1:S9N
8	H5N1:HA1:L9F	8	H5N1:HA1:L9F
9	H5N1:HA1:L1Q	9	H5N1:HA1:L1Q
10	H5N1:HA1:P1S	10	H5N1:HA1:P1S
11	H5N1:HA1:E1D	11	H5N1:HA1:E1D

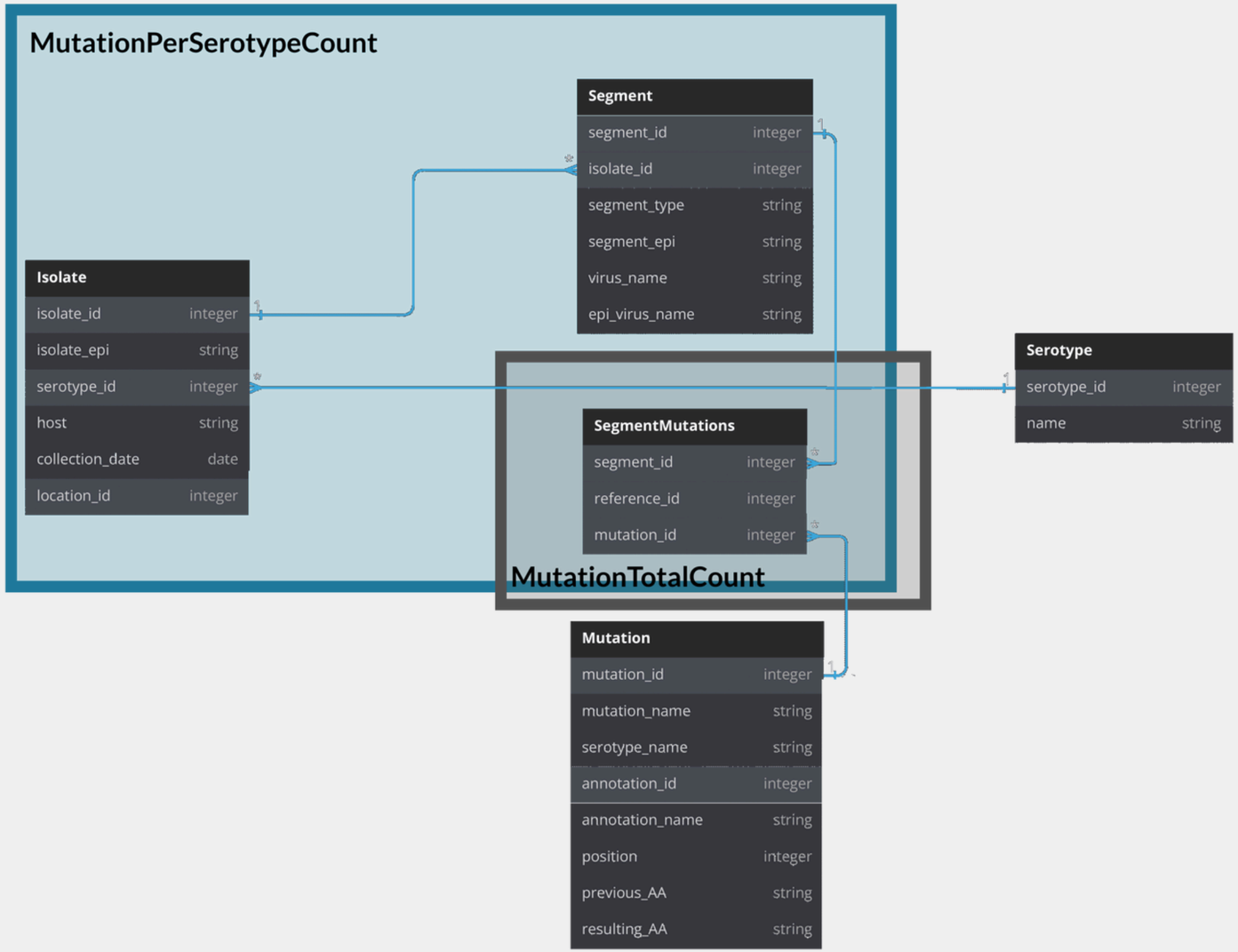
QUERY 3: GET ALL MUTATIONS ASSOCIATED WITH ONE SEROTYPE AND NOT OTHERS



QUERY 3: GET ALL MUTATIONS ASSOCIATED WITH ONE SEROTYPE AND NOT OTHERS

```
SELECT sm.mutation_id, iso.serotype_id
FROM SegmentMutations sm
JOIN Segment s ON sm.segment_id = s.segment_id
JOIN Isolate iso ON s.isolate_id = iso.isolate_epi
WHERE sm.mutation_id NOT IN
    (SELECT sm.mutation_id FROM SegmentMutations sm2
     JOIN Segment s2 ON sm2.segment_id = s2.segment_id
     JOIN Isolate iso2 ON s2.isolate_id = iso2.isolate_epi
     WHERE iso2.serotype_id <> iso.serotype_id)
```

QUERY 4: GET ALL MUTATIONS ASSOCIATED 95% OF THE TIMES WITH ONLY ONE SEROTYPE



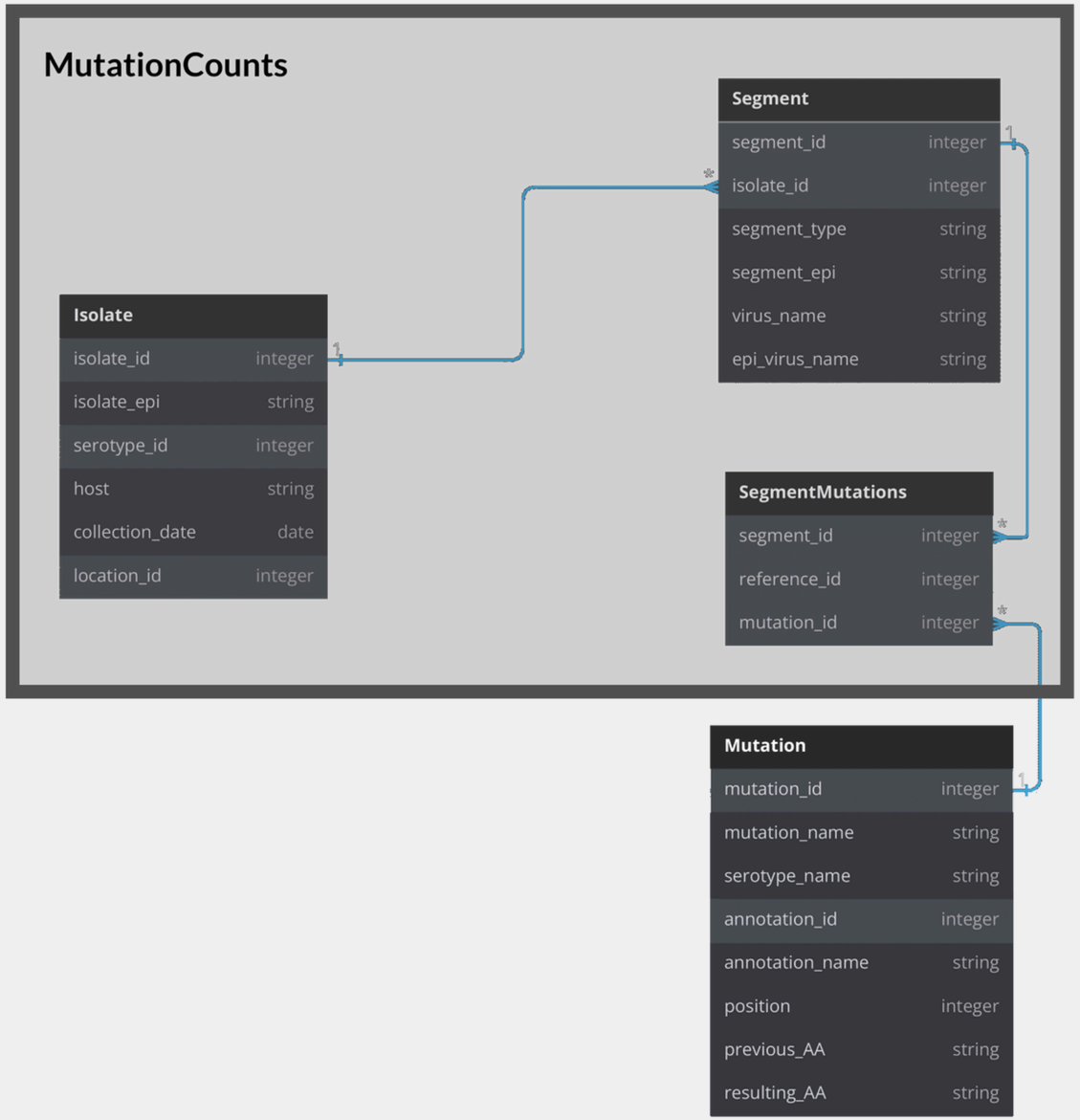
QUERY 4: GET ALL MUTATIONS ASSOCIATED 95% OF THE TIMES WITH ONLY ONE SEROTYPE

```
WITH MutationPerSerotypeCount AS (
    SELECT sm.mutation_id, iso.serotype_id, COUNT(*) AS mutation_count
    FROM SegmentMutations sm
    JOIN Segment s ON sm.segment_id = s.segment_id
    JOIN Isolate iso ON s.isolate_id = iso.isolate_epi
    GROUP BY sm.mutation_id, iso.serotype_id),

MutationTotalCount AS (
    SELECT sm.mutation_id as mutation_id, COUNT(*) AS total_mutation_count
    FROM SegmentMutations sm
    GROUP BY sm.mutation_id)

SELECT m.mutation_name, mpc.mutation_id, sero.name AS serotype_name, mpc.mutation_count,
    mpc.mutation_count * 100.0 / mtc.total_mutation_count AS mutation_percentage
FROM MutationPerSerotypeCount mpc
JOIN MutationTotalCount mtc ON mpc.mutation_id = mtc.mutation_id
JOIN Mutation m ON m.mutation_id = mpc.mutation_id
JOIN Serotype sero ON sero.serotype_id = mpc.serotype_id
WHERE mpc.mutation_count * 100.0 / mtc.total_mutation_count >= 95
```

```sql
WITH MutationPerSerotypeCount AS (
    SELECT sm.mutation_id, iso.serotype_id, COUNT(*) AS mutation_count
    FROM SegmentMutations sm
    JOIN Segment s ON sm.segment_id = s.segment_id
    JOIN Isolate iso ON s.isolate_id = iso.isolate_epi
    GROUP BY sm.mutation_id, iso.serotype_id),
MutationTotalCount AS (
    SELECT sm.mutation_id as mutation_id, COUNT(*) AS total_mutation_count
    FROM SegmentMutations sm
    GROUP BY sm.mutation_id)
SELECT m.mutation_name, mpc.mutation_id, sero.name AS serotype_name, mpc.mutation_count,
    mpc.mutation_count * 100.0 / mtc.total_mutation_count AS mutation_percentage
FROM MutationPerSerotypeCount mpc
JOIN MutationTotalCount mtc ON mpc.mutation_id = mtc.mutation_id
JOIN Mutation m ON m.mutation_id = mpc.mutation_id
JOIN Serotype sero ON sero.serotype_id = mpc.serotype_id
WHERE mpc.mutation_count * 100.0 / mtc.total_mutation_count >= 95
```

	mutation_name	mutation_id	serotype_name	mutation_count	mutation_percentage
1	H5N1:HA1:K3R	1	H5N1	23	100.0
2	H5N1:HA1:K5R	2	H5N1	5	100.0
3	H5N1:HA1:E7N	3	H5N1	23	100.0
4	H5N1:HA1:R8K	4	H5N1	23	100.0
5	H5N1:HA1:N8S	5	H5N1	3	100.0

QUERY 5A: GET ALL MUTATIONS BY % OF HUMAN HOSTS



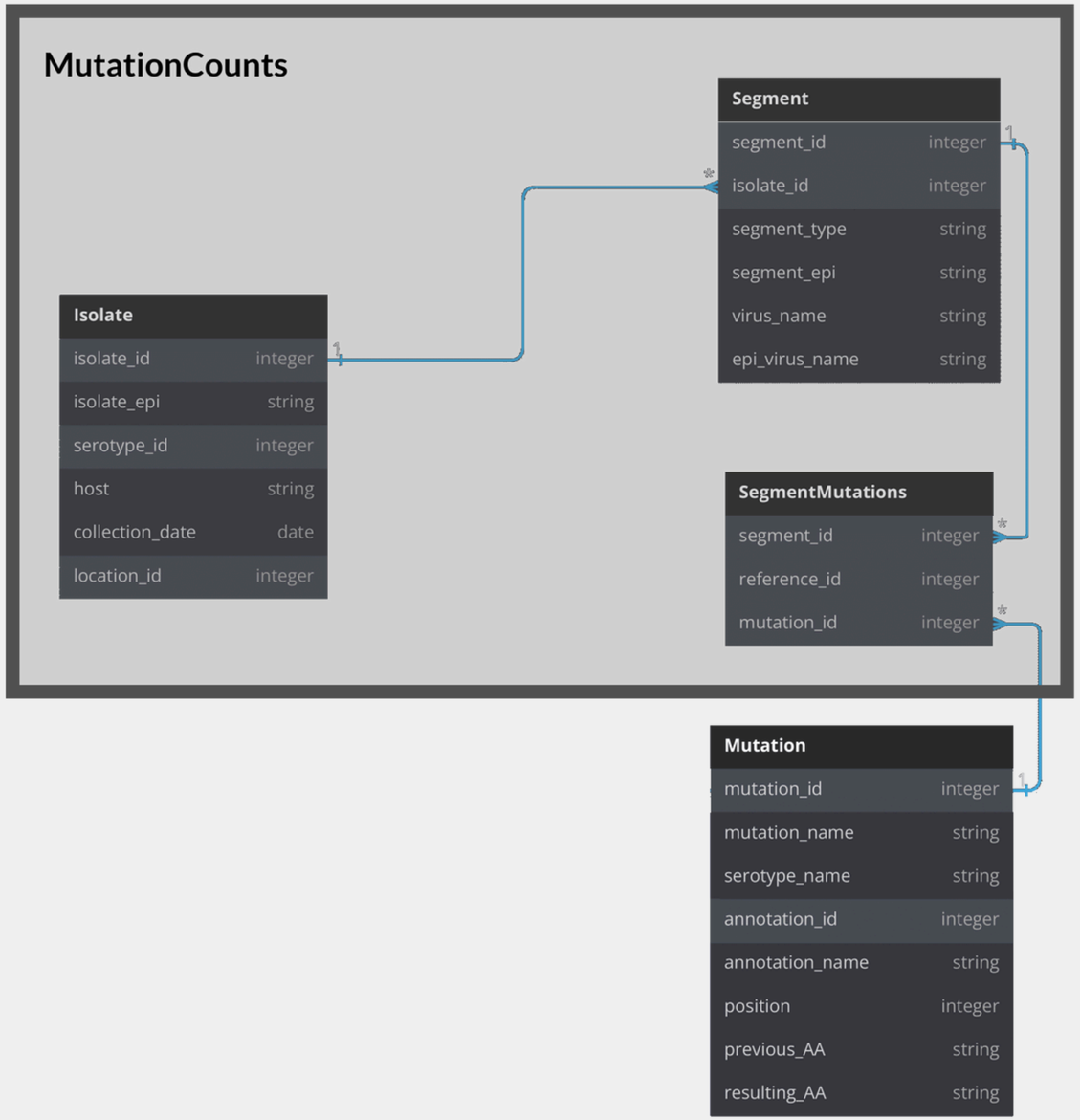
QUERY 5A: GET ALL MUTATIONS BY % HUMAN HOSTS

```
CREATE VIEW MutationCounts AS
SELECT sm.mutation_id, COUNT(*) AS total_count,
SUM(CASE WHEN iso.host = 'Human' THEN 1 ELSE 0 END) AS human_count
FROM SegmentMutations sm JOIN Segment seg ON sm.segment_id =
seg.segment_id
JOIN Isolate iso ON seg.isolate_id = iso.isolate_epi
GROUP BY sm.mutation_id

SELECT m.mutation_name, mutation_id, total_count, human_count,
human_count * 100.0 / total_count AS human_percentage
FROM MutationCounts mc
JOIN Mutation m on m.mutation_id = mc.mutation_id
ORDER BY human_count * 100.0 / total_count DESC
```

```sql
SELECT m.mutation_name, mc.mutation_id, total_count, human_count, human_count * 100.0 / total_count AS human_percentage
FROM MutationCounts mc
JOIN Mutation m on m.mutation_id = mc.mutation_id
ORDER BY human_count * 100.0 / total_count DESC
```

	mutation_name	mutation_id	total_count	human_count	human_percentage
1	H5N1:HA1:K5R	2	5	5	100.0
2	H5N1:HA1:N8S	5	3	3	100.0
3	H5N1:HA1:E1D	11	3	3	100.0
4	H5N1:HA1:L1S	13	3	3	100.0
5	H5N1:HA1:A1S	14	5	5	100.0
6	H5N1:HA1:Q1L	15	3	3	100.0
7	H5N1:HA1:A1R	16	2	2	100.0
8	H5N1:HA1:S2P	27	3	3	100.0
9	H5N1:HA1:V2M	35	5	5	100.0
10	H5N1:HA2:D5N	38	5	5	100.0
11	H5N1:HA1:N1S	44	5	5	100.0
12	H5N1:HA1:A1K	45	5	5	100.0
13	H5N1:HA1:S3A	48	1	1	100.0
14	H5N1:HA2:K3Q	49	3	3	100.0
15	H5N1:HA1:A8V	52	2	2	100.0
16	H5N1:HA1:S9D	53	3	3	100.0
17	H5N1:HA1:P1L	54	2	2	100.0

QUERY 5B: GET ALL MUTATIONS WHERE HUMAN HOSTS % >= 95%



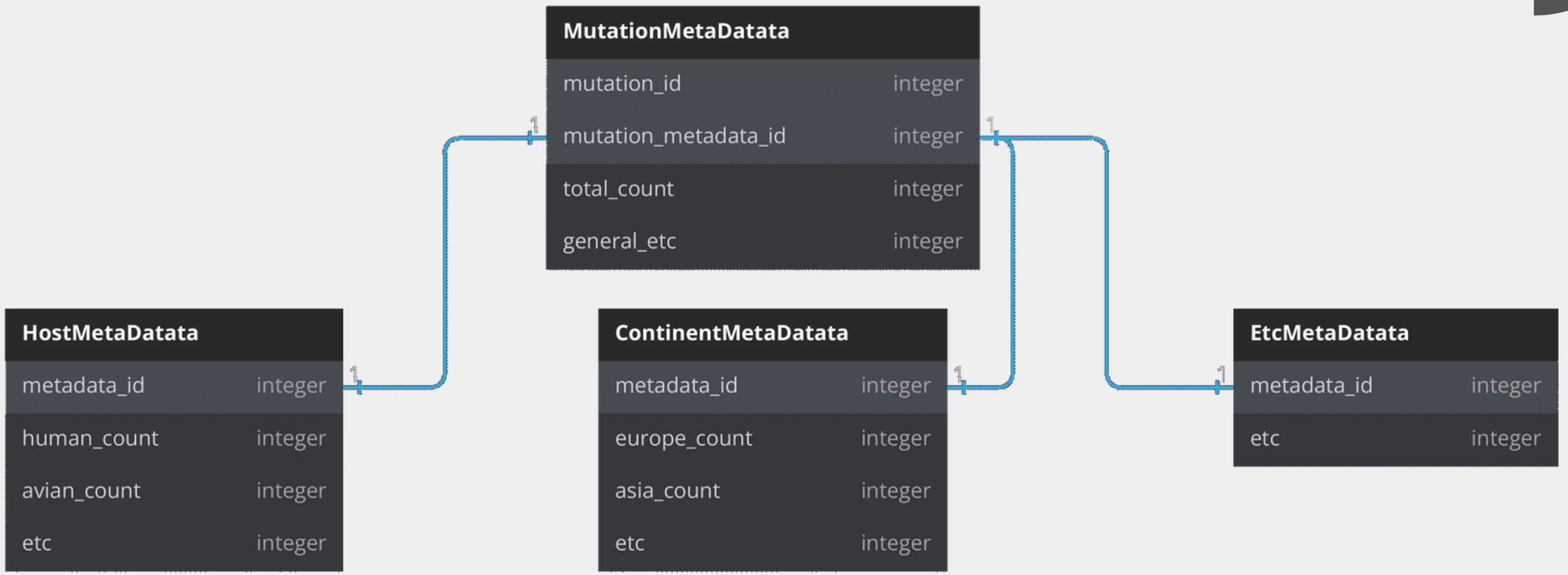
QUERY 5B: GET ALL MUTATIONS WHERE HUMAN HOSTS % >= 95%

```
CREATE VIEW MutationCounts AS
SELECT sm.mutation_id, COUNT(*) AS total_count,
SUM(CASE WHEN iso.host = 'Human' THEN 1 ELSE 0 END) AS human_count
FROM SegmentMutations sm JOIN Segment seg ON sm.segment_id =
seg.segment_id
JOIN Isolate iso ON seg.isolate_id = iso.isolate_epi
GROUP BY sm.mutation_id

SELECT m.mutation_name, mc.mutation_id, total_count, human_count,
human_count * 100.0 / total_count AS human_percentage
FROM MutationCounts mc
JOIN Mutation m on m.mutation_id = mc.mutation_id
WHERE human_count * 100.0 / total_count >= 95.0
```

```sql
SELECT m.mutation_name, mc.mutation_id, total_count, human_count, human_count * 100.0 / total_count AS human_percentage
FROM MutationCounts mc
JOIN Mutation m on m.mutation_id = mc.mutation_id
WHERE human_count * 100.0 / total_count >= 95.0
```

	mutation_name	mutation_id	total_count	human_count	human_percentage
1	H5N1:HA1:K5R	2	5	5	100.0
2	H5N1:HA1:N8S	5	3	3	100.0
3	H5N1:HA1:E1D	11	3	3	100.0
4	H5N1:HA1:L1S	13	3	3	100.0
5	H5N1:HA1:A1S	14	5	5	100.0
6	H5N1:HA1:Q1L	15	3	3	100.0
7	H5N1:HA1:A1R	16	2	2	100.0
8	H5N1:HA1:S2P	27	3	3	100.0
9	H5N1:HA1:V2M	35	5	5	100.0
10	H5N1:HA2:D5N	38	5	5	100.0
11	H5N1:HA1:N1S	44	5	5	100.0
12	H5N1:HA1:A1K	45	5	5	100.0
13	H5N1:HA1:S3A	48	1	1	100.0
14	H5N1:HA2:K3Q	49	3	3	100.0
15	H5N1:HA1:A8V	52	2	2	100.0
16	H5N1:HA1:S9D	53	3	3	100.0
17	H5N1:HA1:P1L	54	2	2	100.0

QUERY 5C: GET ALL MUTATIONS BY % HUMAN HOSTS WHERE >= 95%



QUERY 5C: GET ALL MUTATIONS BY % HUMAN HOSTS WHERE >= 95%

```
SELECT mutation_id,
hmd.human_count / md.total_count AS human_percentage
FROM MutationMetaData md
JOIN HostMetaData hmd ON md.id = hmd.medatata_id
WHERE md.total_count > 0
AND hmd.human_count / md.total_count >= 0.95
ORDER BY hmd.human_count * 100.0 / md.total_count
DESC
```

QUERY 6, 7, 8

Mutation	
mutation_id	integer
mutation_name	string
serotype_name	string
annotation_id	integer
annotation_name	string
position	integer
previous_AA	string
resulting_AA	string

QUERY 6: GET THE MOST COMMON AA MUTATIONS

```
CREATE VIEW MostCommonAAMutation AS
SELECT previous_AA, resulting_AA, COUNT(*) AS count
FROM Mutation
GROUP BY previous_AA, resulting_AA

SELECT * FROM MostCommonAAMutation
ORDER BY count DESC
```

	previous_AA	resulting_AA	count
1	K	R	23
2	R	K	16
3	V	I	15
4	I	V	14
5	T	A	13
6	D	N	12
7	S	N	12
8	N	S	11
9	A	T	10
10	A	V	9
11	E	G	9
12	I	L	9
13	F	K	8

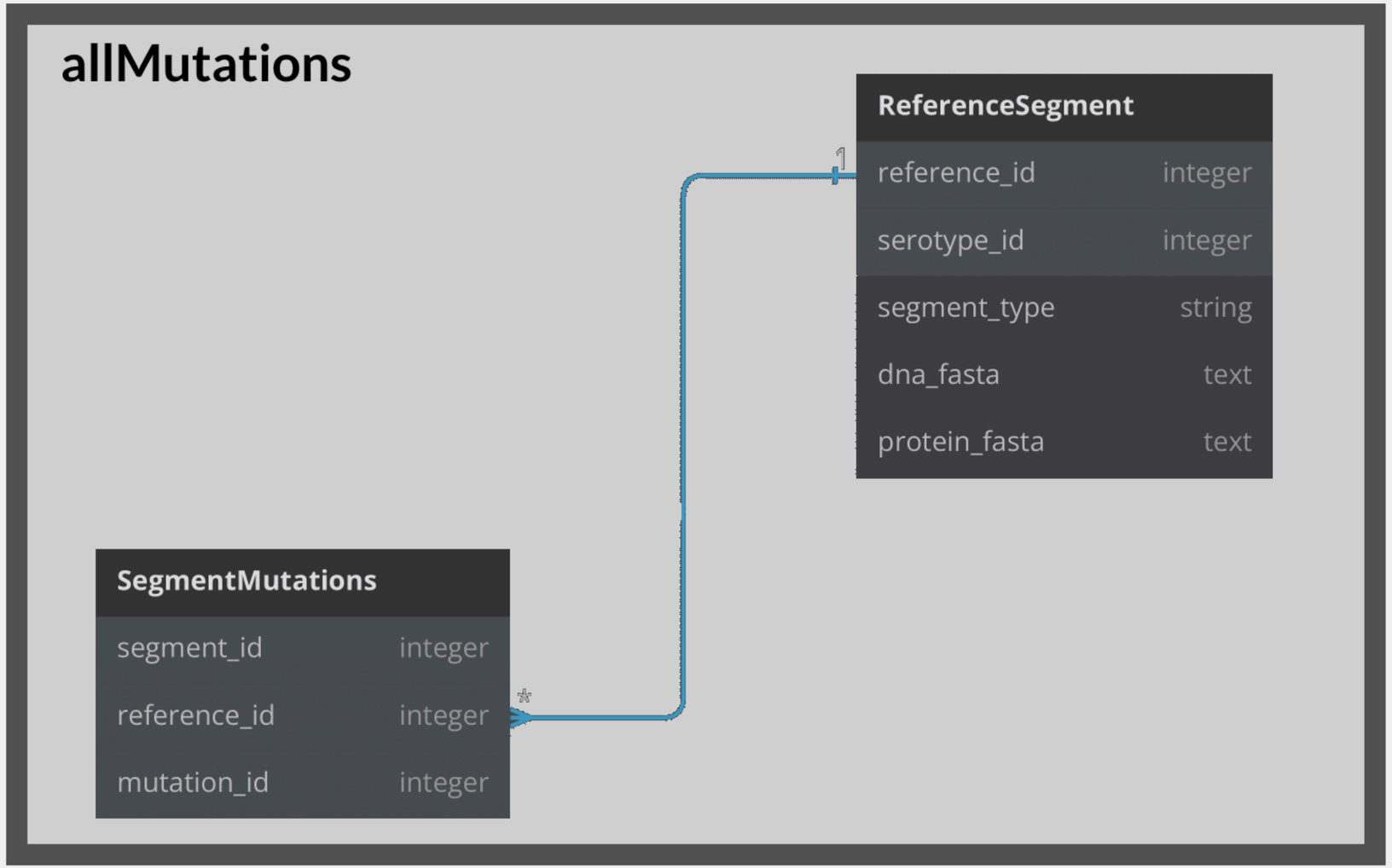
QUERY 7: GET THE MOST COMMON AA MUTATIONS PER ANNOTATION

```
CREATE VIEW MostCommonAnnotationMutation AS
SELECT previous_AA, resulting_AA, COUNT(*) AS count
FROM Mutation
GROUP BY annotation_name, previous_AA, resulting_AA

SELECT * FROM MostCommonAnnotationMutation
ORDER BY count DESC
```

QUERY 8: GET THE MOST COMMON AA MUTATIONS COMBINATION OF ANNOTATION AND SEROTYPE

```
CREATE VIEW MostCommonSerotypeAnnotationMutation AS
SELECT previous_AA, resulting_AA, COUNT(*) AS count
FROM Mutation
GROUP BY serotype_id, annotation_name, previous_AA, resulting_AA

SELECT * FROM MostCommonSerotypeAnnotationMutation
ORDER BY count DESC
```

QUERY 9: FOR EACH MUTATION GET THE NUMBER OF SEGMENTS THAT EXHIBIT IT, AND THE PERCENTAGE OVER THE TOTAL NUMBER OF SEGMENTS



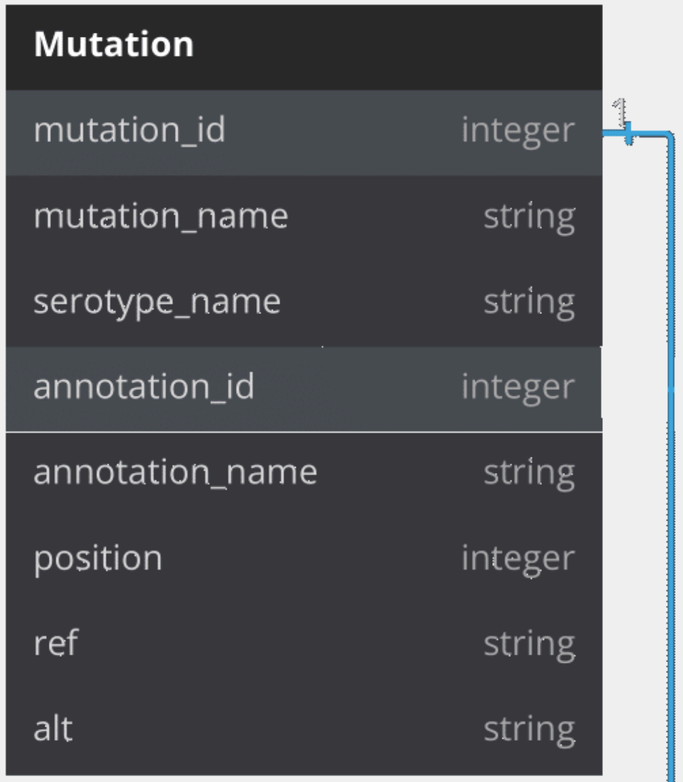
QUERY 9: FOR EACH MUTATION GET THE NUMBER OF SEGMENTS THAT EXHIBIT IT, AND THE PERCENTAGE OVER THE TOTAL NUMBER OF SEGMENTS

```
SELECT m.mutation_name, sm.mutation_id, sm.reference_id, COUNT(*) as count,
    COUNT(*) / (SELECT COUNT(DISTINCT sm.segment_id) FROM SegmentMutations WHERE
reference_id = sm.reference_id) as percentage
FROM SegmentMutations sm
JOIN Mutation m ON m.mutation_id = sm.mutation_id
GROUP BY sm.mutation_id, sm.reference_id
ORDER BY count DESC
```

```sql
SELECT m.mutation_name, sm.mutation_id, sm.reference_id, COUNT(*) as count,
      COUNT(*) * 1.0 / (SELECT COUNT(*) FROM SegmentMutations WHERE reference_id = sm.reference_id) as percentage
FROM SegmentMutations sm
JOIN Mutation m ON m.mutation_id = sm.mutation_id
GROUP BY sm.mutation_id, sm.reference_id
ORDER BY count DESC
```

	mutation_name	mutation_id	reference_id	count	percentage
1	H5N1:HA1:K3R	1	3	23	0.0153027278775782
2	H5N1:HA1:E7N	3	3	23	0.0153027278775782
3	H5N1:HA1:R8K	4	3	23	0.0153027278775782
4	H5N1:HA1:L9F	8	3	23	0.0153027278775782
5	H5N1:HA1:P1S	10	3	23	0.0153027278775782
6	H5N1:HA1:D1S	17	3	23	0.0153027278775782
7	H5N1:HA1:S1P	21	3	23	0.0153027278775782
8	H5N1:HA1:N1D	22	3	23	0.0153027278775782
9	H5N1:HA1:E1A	23	3	23	0.0153027278775782
10	H5N1:HA1:N1K	24	3	23	0.0153027278775782
11	H5N1:HA1:K1Q	25	3	23	0.0153027278775782
12	H5N1:HA1:K2E	26	3	23	0.0153027278775782
13	H5N1:HA1:D2N	31	3	23	0.0153027278775782
14	H5N1:HA1:H2N	32	3	23	0.0153027278775782
15	H5N1:HA1:G2E	33	3	23	0.0153027278775782
16	H5N1:HA1:V2L	34	3	23	0.0153027278775782
17	H5N1:HA1:S3T	36	3	23	0.0153027278775782

QUERY 10: RETRIEVE ALL MARKERS ASSOCIATED WITH A PARTICULAR MUTATION



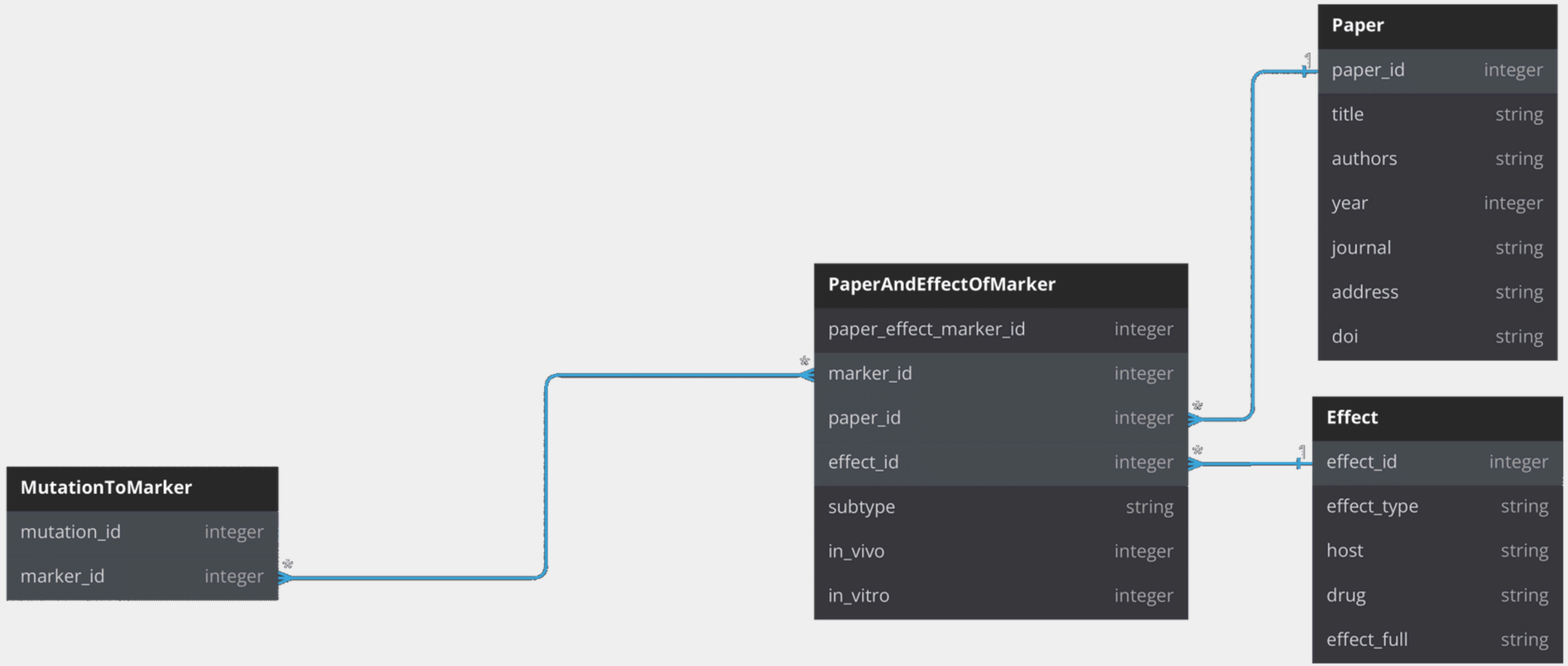
QUERY 10: RETRIEVE ALL MARKERS ASSOCIATED TO GROUPS THAT CONTAIN A PARTICULAR MUTATION

```
SELECT marker_id FROM MutationsToMarker mtm
JOIN Mutation mut ON mut.mutation_id = mtm.mutation_id
WHERE mut.mutation_name = "H5N1:M2:L26F"
```

QUERY 10B: RETRIEVE THE MARKER (IF ANY) ASSOCIATED TO A GROUP OF MUTATIONS

```
WITH mutation_group AS
    (SELECT mutation_id
     FROM Mutation
     WHERE mutation_name IN (???)),

SELECT marker_id FROM MutationsToMarker
WHERE mutation_id IN (SELECT mutation_id  FROM mutation_group)
GROUP BY marker_id
HAVING COUNT(CASE WHEN mutation_id  IN (SELECT mutation_id  FROM mutation_group ) THEN 1
END) = COUNT(*)
AND COUNT(CASE WHEN mutation_id IN (SELECT mutation_id FROM mutation_group ) THEN 1 END)
= (SELECT COUNT(*) FROM mutation_group)
```

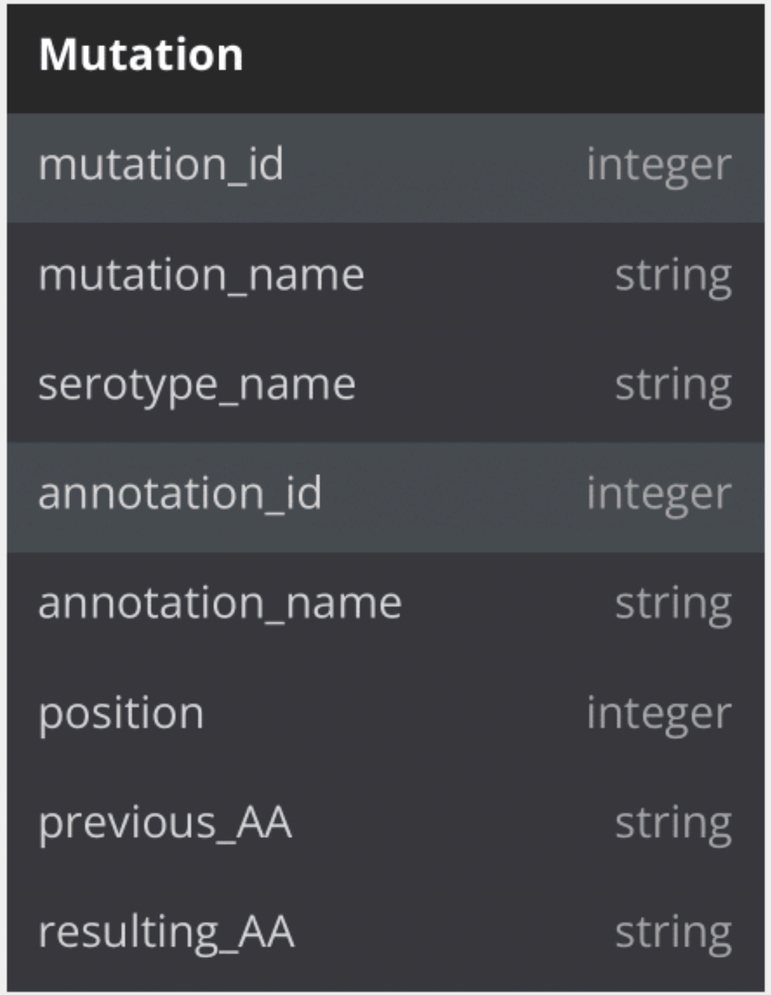
QUERY 11: GET ALL EFFECTS ASSOCIATED WITH A SPECIFIC MUTATION



QUERY 11: GET ALL EFFECTS ASSOCIATED WITH A SPECIFIC MUTATION

```
SELECT * FROM Effect e
WHERE e.effect_id in (
    SELECT pem.effect_id
    FROM MutationsToMarker mtm
    JOIN PaperAndEffectOfMarker pem ON mtm.marker_id = pem.marker_id
    WHERE mtm.mutation_id = 6501)
```

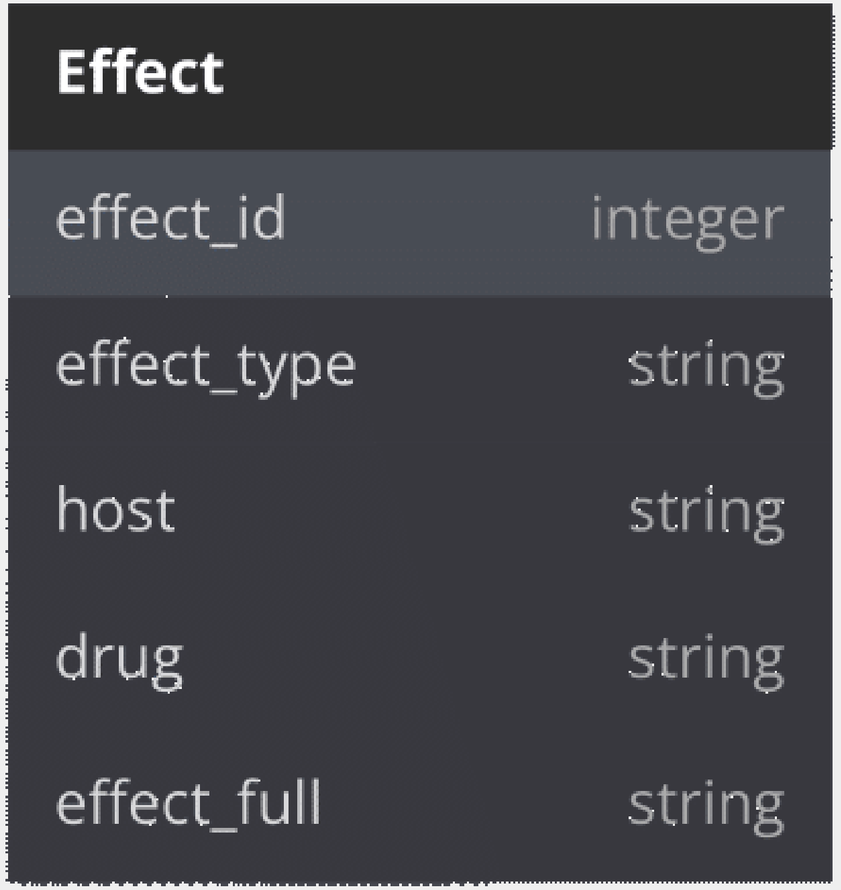
QUERY 12: IDENTIFY MUTATIONS AT A SPECIFIC POSITION.



QUERY 12: IDENTIFY MUTATIONS AT A SPECIFIC POSITION.

```
SELECT mutation_name
FROM Mutation
WHERE position > 0
AND position < 1000
```

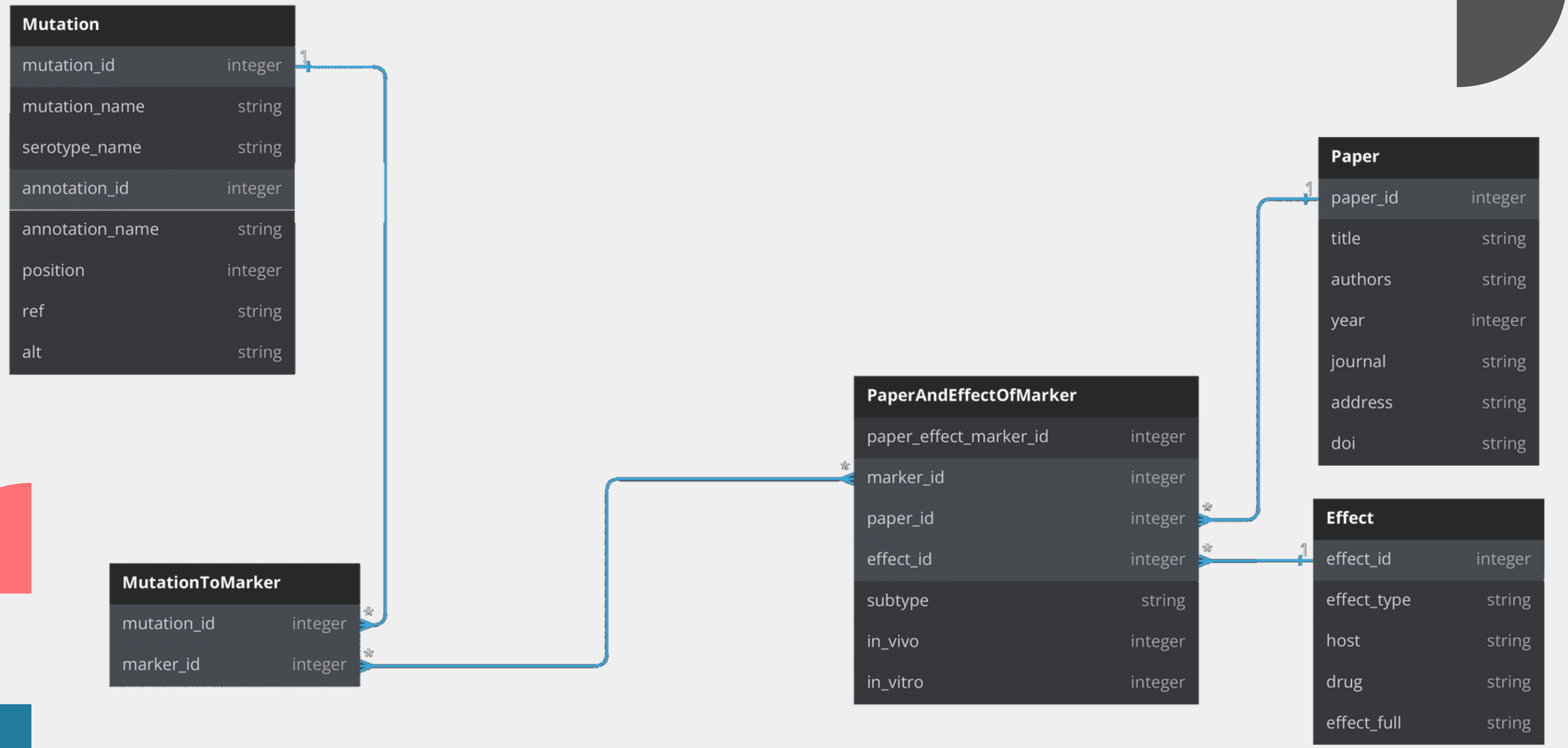
QUERY 13: RETRIEVE ALL EFFECTS BY SPECIFIC TYPE



QUERY 13: RETRIEVE ALL EFFECTS BY SPECIFIC TYPE

```
SELECT effect_full
from Effect
WHERE effect_type LIKE '%Increased%'
OR effect_type LIKE '%Decreased%'
```

QUERY 14: IDENTIFY MUTATIONS ASSOCIATED WITH A PARTICULAR EFFECT TYPE



QUERY 14: IDENTIFY MUTATIONS ASSOCIATED WITH A PARTICULAR EFFECT TYPE

```
SELECT DISTINCT mut.mutation_name
FROM Mutation mut
JOIN MutationsToMarker mtm ON mut.mutation_id = mtm.mutation_id
JOIN PaperAndEffectOfMarker pem ON mtm.marker_id = pem.marker_id
JOIN Effect e ON pem.effect_id = e.effect_id
WHERE e.effect_type LIKE '%Increased%'
```

TABLE OF CONTENT

NEXT STEPS

ARCHITECTURE

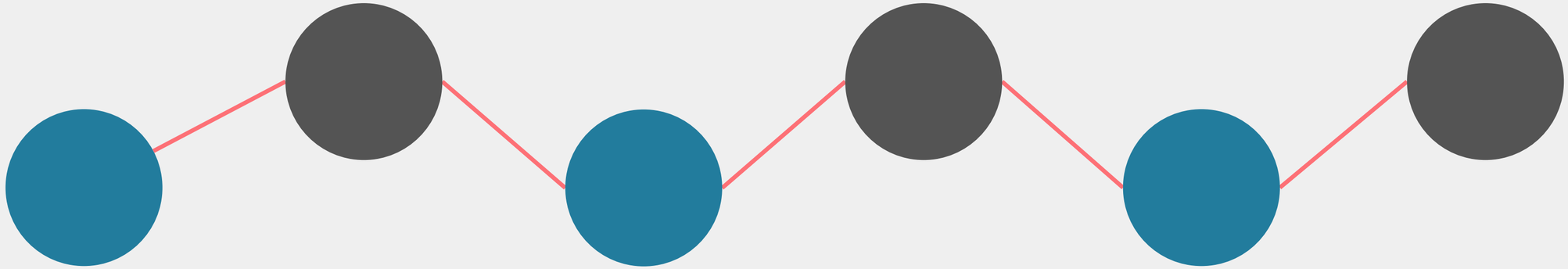
DESIGN AND CONCEPTUALIZE THE ARCHITECTURE OF THE ANALYTICAL FRAMEWORK, ENSURING SCALABILITY, EFFICIENCY

DOCKER

IMPLEMENT DOCKER CONTAINERS TO DEPLOY THE PROJECT, ENSURING CONSISTENCY ACROSS DIFFERENT ENVIRONMENTS

UPDATES

CONTINUOUSLY REFINE AND UPDATE THE WEBSITE, INCORPORATING NEW DATA AND INSIGHTS OVER TIME



NEW QUERIES

EXPAND THE SCOPE OF DATA COLLECTION BY FORMULATING AND EXECUTING REFINED QUERIES TO GATHER ADDITIONAL PERTINENT DATA ON FLU MUTATIONS

DEVELOPMENT

DEVELOPMENT PHASE, TRANSLATING ARCHITECTURAL BLUEPRINTS INTO FUNCTIONAL CODE, ITERATIVELY REFINING ALGORITHMS AND FUNCTIONALITIES

PUBLIC WEBSITE

PUBLIC-FACING WEBSITE TO PUBLISH RESEARCH FINDINGS, PROVIDING A USER-FRIENDLY INTERFACE FOR ACCESSING VISUALIZATIONS

THANK YOU

ANY QUESTION?