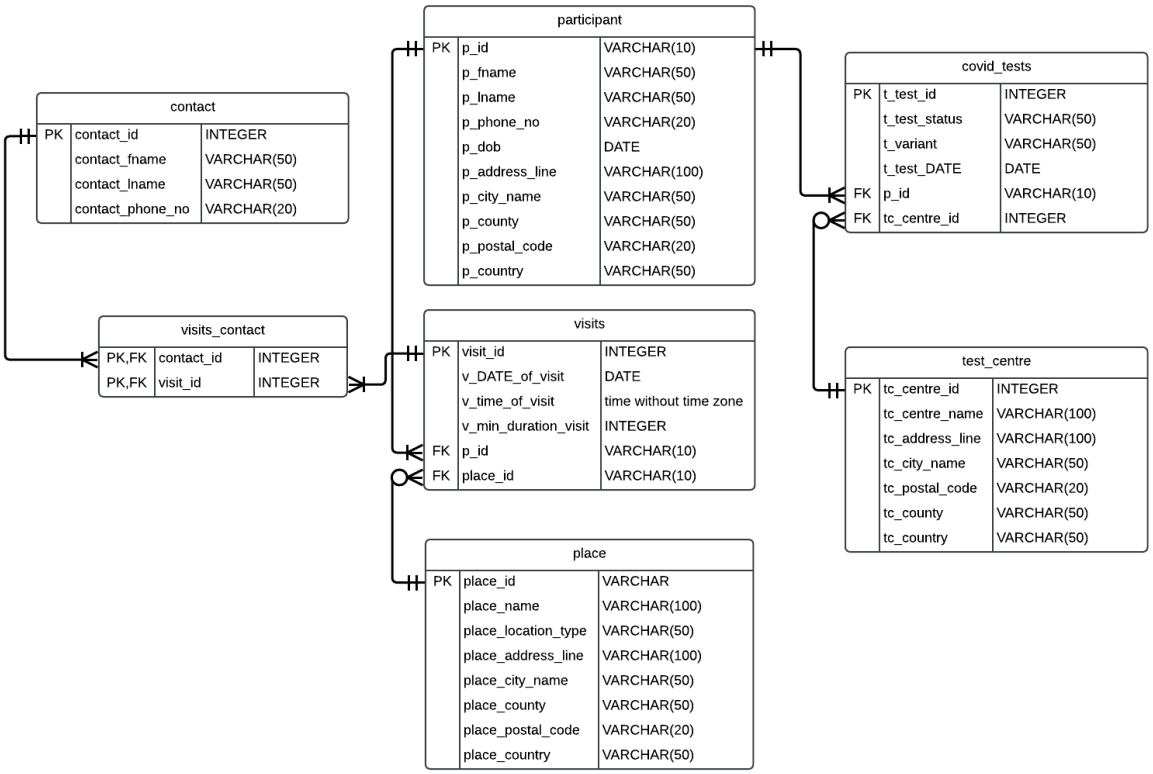
Task 1: ERD with a short description.



The Entity-Relationship Diagram (ERD) illustrates the structure of our database with seven entities, their attributes, and their relationships. The 'participant' table stores comprehensive details about the individuals under our observation. Its primary key, 'p_id', uniquely identifies each person. This table establishes one-to-many relationships with both the 'covid_tests' and 'visits' tables through the 'p_id' attribute.

Moving to the 'covid_tests' table, it's linked to the 'test_centre' table via the 'tc_centre_id'. This connection forms a relationship where a test_centre can conduct multiple tests, indicating a zero-to-many relationship between 'test_centre' and 'covid_tests'.

The 'visits' table acts as a bridge between 'participants' and 'places' tables. It avoids a direct many-to-many relationship between individuals and locations. 'Visits' describe individual instances of participants' trips or outings around the test period, connecting participants and places they've been to. Simultaneously, the 'place' table shows detailed location information about these visited places.

Another vital table, 'contacts', catalogues individuals our participants have encountered recently. It's pivotal for contact tracing and adopting necessary preventive measures. Finally, the 'visits_contact' table acts as an intermediary between 'visits' and 'contacts', forging connections between participant outings and their associated contacts for efficient tracking and preventive strategies.

Task 2: Create the database using PostgreSQL, which can be created on your VM or another platform. Work not created using PostgreSQL will receive no marks. Fill your database with dummy data eg 20 records per table.

```
cov19_tracker=# \d
                    List of relations
 Schema |             Name              |   Type   |   Owner
--------+-------------------------------+----------+-----------
 public | contact                       | table    | up2225817
 public | contact_contact_id_seq        | sequence | up2225817
 public | covid_tests                   | table    | up2225817
 public | covid_tests_t_test_id_seq     | sequence | up2225817
 public | participant                   | table    | up2225817
 public | place                         | table    | up2225817
 public | test_centre                   | table    | up2225817
 public | test_centre_tc_centre_id_seq  | sequence | up2225817
 public | visits                        | table    | up2225817
 public | visits_contact                | table    | up2225817
 public | visits_contact_contact_id_seq | sequence | up2225817
 public | visits_contact_visit_id_seq   | sequence | up2225817
 public | visits_visit_id_seq           | sequence | up2225817
(13 rows)
```

Task 3: Write SQL queries to show the following:

a. Test report, showing all the test results for a single participant identified by date of birth, family name and postcode.

```
cov19_tracker=# -- 1. Test report, showing all the test results for a single participant identified by date of birth,
-- family name and postcode
SELECT PART.P_DOB,
       PART.P_LNAME,
       PART.P_POSTAL_CODE,
       CT.*
FROM PARTICIPANT PART
JOIN COVID_TESTS CT ON PART.P_ID = CT.P_ID
ORDER BY CT.P_ID, CT.T_TEST_DATE, CT.TC_CENTRE_ID;
cov19_tracker=#
```

p_dob	p_lname	p_postal_code	t_test_id	t_test_status	t_variant	t_test_date	p_id	tc_centre_id
1979-11-11	Battle	NL49 2KS	550007	Positive	Omicron	2019-07-13	PID001	6
1979-11-11	Battle	NL49 2KS	550002	Negative	NA	2020-02-08	PID001	2
1946-08-08	Lamb	VI8 1XN	550017	Negative	NA	2020-03-25	PID002	5
1946-08-08	Lamb	VI8 1XN	550005	Negative	NA	2020-03-31	PID002	2
1946-08-08	Lamb	VI8 1XN	550014	Positive	Delta	2020-04-25	PID002	5
1946-08-08	Lamb	VI8 1XN	550039	Negative	NA	2023-11-28	PID002	7
2013-04-11	Langley	E7K 7BZ	550019	Positive	Omicron	2020-05-13	PID003	4
2013-04-11	Langley	E7K 7BZ	550020	Positive	Delta	2020-05-16	PID003	7
2013-04-11	Langley	E7K 7BZ	550016	Positive	Omicron	2020-06-22	PID003	1
2013-04-11	Langley	E7K 7BZ	550041	Positive	Delta	2022-12-31	PID003	1
2013-04-11	Langley	E7K 7BZ	550033	Negative	NA	2023-12-16	PID003	3
2003-12-18	Hood	B18 0EI	550029	Negative	NA	2020-11-28	PID004	7
2003-12-18	Hood	B18 0EI	550047	Negative	NA	2023-02-20	PID004	7
2003-12-18	Hood	B18 0EI	550035	Positive	Beta	2023-10-31	PID004	2
2000-07-12	Gomez	N3J 6DM	550052	Positive	Delta	2023-08-13	PID005	8
2020-05-25	Webb	K87 4XL	550023	Positive	Alpha	2019-12-31	PID006	1
2020-05-25	Webb	K87 4XL	550042	Positive	Alpha	2022-12-07	PID006	2
1999-12-20	Michael	N4 2OS	550026	Positive	Omicron	2020-06-28	PID007	4
1999-12-20	Michael	N4 2OS	550021	Positive	Alpha	2020-10-19	PID007	2
1999-12-20	Michael	N4 2OS	550006	Negative	NA	2020-12-10	PID007	5
1964-01-02	Solomon	LT3 2YE	550049	Positive	Omicron	2023-05-25	PID008	9
1967-07-20	ONeill	PK3M 1FE	550004	Positive	Omicron	2019-03-24	PID009	6
1967-07-20	ONeill	PK3M 1FE	550018	Negative	NA	2020-05-04	PID009	5
1967-07-20	ONeill	PK3M 1FE	550043	Negative	NA	2023-05-18	PID009	3
1980-05-13	Gordon	V5D 7TK	550030	Positive	Beta	2019-08-11	PID010	9
1980-05-13	Gordon	V5D 7TK	550040	Positive	Beta	2023-08-11	PID010	9
1952-12-18	Nieves	KB7 5TF	550031	Negative	NA	2023-08-10	PID011	9
1987-07-18	Petty	MK3 6NI	550003	Positive	Omicron	2020-01-29	PID012	4
1987-07-18	Petty	MK3 6NI	550044	Positive	Delta	2023-03-19	PID012	4
1955-07-24	Cote	TR8 5NH	550022	Negative	NA	2019-08-10	PID013	3
1988-01-03	Lindsey	XU3 4SY	550010	Positive	Omicron	2020-06-20	PID014	7
1988-01-03	Lindsey	XU3 4SY	550025	Negative	NA	2020-10-31	PID014	2
2000-09-05	Dorsey	OE7K 8EV	550009	Positive	Omicron	2020-08-29	PID015	7
2000-09-05	Dorsey	OE7K 8EV	550045	Positive	Alpha	2023-10-01	PID015	5
1936-12-08	Price	N05 4GX	550012	Positive	Omicron	2020-03-02	PID016	4
1936-12-08	Price	N05 4GX	550048	Positive	Alpha	2023-01-19	PID016	8
1936-12-08	Price	N05 4GX	550050	Negative	NA	2023-06-12	PID016	10
1936-12-08	Price	N05 4GX	550032	Positive	Beta	2023-12-31	PID016	1
2017-05-10	Petersen	JJ6 3MO	550015	Positive	Beta	2019-04-14	PID017	7
2017-05-10	Petersen	JJ6 3MO	550036	Positive	Omicron	2023-06-28	PID017	4
1983-06-01	Conley	C2 8TA	550028	Positive	Alpha	2020-10-16	PID018	4
1983-06-01	Conley	C2 8TA	550046	Negative	NA	2022-12-01	PID018	6
1983-06-01	Conley	C2 8TA	550038	Positive	Alpha	2023-10-16	PID018	4
1960-03-30	Walton	O3S 4QK	550013	Negative	NA	2019-03-20	PID019	7
1960-03-30	Walton	O3S 4QK	550011	Positive	Delta	2019-10-01	PID019	2
1960-03-30	Walton	O3S 4QK	550024	Negative	NA	2020-12-16	PID019	3
2008-04-02	Rivas	QS26 2QG	550027	Negative	NA	2019-05-11	PID020	2

:

b. Virus variant distribution over the last 12 months.

```
cov19_tracker=# -- 2. Virus variant distribution over the last 12 months
SELECT T_VARIANT,
        COUNT(*) AS VARIANT_COUNT
FROM COVID_TESTS
WHERE T_TEST_DATE >= CURRENT_DATE - INTERVAL '12 months'
GROUP BY T_VARIANT;
 t_variant | variant_count
-----------+---------------
 Alpha     |             3
 Delta     |             3
 Omicron   |             3
 Beta      |             3
 NA        |             7
(5 rows)
```

c. Number of positive tests over the last 12 months per country i.e. England, Scotland, Wales and Northern Ireland.

```
cov19_tracker=# select tc.tc_country,
count(*) as covid_positive_count
from covid_tests ct join test_centre tc on ct.tc_centre_id=tc.tc_centre_id where t_test_status >= 'Positive'
group by tc.tc_country order by covid_positive_count desc;
   tc_country    | covid_positive_count
-----------------+----------------------
 Scotland        |                   14
 Wales           |                    8
 England         |                    5
 Nothern Ireland |                    4
(4 rows)
```

TASK 4: Short report.

NoSQL, short for "Not only SQL," diverges from the traditional relational database structure by storing data in non-tabular formats. While the roots of NoSQL databases trace back to the 1960s, their widespread adoption and optimization occurred in the early 2000s, revolutionising database development practices (Hiteshreddy2181, 2023; Ashtari, 2022; MongoDB, nd).
This database paradigm gained prominence notably in major internet companies dealing with massive data volumes. Unlike conventional RDBMS, NoSQL addressed the performance limitations encountered when handling extensive datasets, mitigating slow system responses. Rather than solely scaling up hardware, NoSQL introduced the concept of scaling out, distributing the database workload across multiple hosts as demand increases (Taylor, 2023). This approach helped alleviate performance bottlenecks and offered a cost-effective solution to handling growing data loads.

Some key features of NoSQL are:

1. Flexibility: Documents in a collection aren't obligated to possess identical field sets or data types. While there can be variations in field data types among documents in a collection, they usually follow a similar structure (MongoDB, nd).
2. Horizontal Scaling: Relational databases scale with effort due to their traditional server-client architecture. In contrast, NoSQL databases offer a serverless, peer-to-peer system across all nodes, allowing straightforward scalability for cloud applications. Sharding in NoSQL enables horizontal scaling by partitioning and distributing data across multiple machines while maintaining its order. Additionally, NoSQL's auto-replication feature ensures high availability by responding with replicated data in case of failure, maintaining a consistent state (Ashtari, 2022).
3. Minimal Downtime: NoSQL databases ensure minimal downtime by employing serverless architectures and maintaining multiple data copies across nodes. In case of node failure, alternative nodes provide access to their data copies, ensuring continuous operations.

Types of NoSQL Include:

1. Key-value pair NoSQL: In this model, data is stored as pairs of keys and associated values, similar to a dictionary or hash table. The simplicity of this structure allows for rapid storage and retrieval of data. These databases accommodate various data types for values, such as strings, JSON, or binary large objects (BLOBs). Key-value stores are highly scalable and efficient for managing large volumes of data. They find extensive use in scenarios like caching, session storage, and e-commerce applications. Examples include Redis, known for its lightning-fast in-memory data storage and retrieval, and Amazon DynamoDB, offering high scalability and low latency for distributed applications.

2. Column-based NoSQL: This type organises data in columns rather than rows, making it suitable for handling queries that require aggregating large amounts of data. Each column can be accessed independently, making column-based stores efficient for analytics and complex querying. The schema-less design of these

databases allows for flexible column addition without altering the entire structure, making them ideal for applications requiring high read and write speeds. Cassandra, known for its ability to handle massive amounts of data across multiple nodes and high availability, and Apache HBase, designed for random, real-time read/write access to big data, are prominent examples.

3. Document-oriented NoSQL: Document databases store data as self-describing JSON or XML documents, offering flexibility without strict schemas. Each document can have a different structure, allowing developers to store related data together. These databases are well-suited for applications where data evolves or requires frequent updates. MongoDB, a popular choice due to its flexibility, scalability, and powerful querying capabilities, CouchDB, known for its distributed architecture and offline functionality, and Amazon SimpleDB, offering simplicity and indexing, are leading examples.

4. Graph-type NoSQL: Graph databases model relationships between entities as nodes and edges, representing networks and complex hierarchical structures efficiently. These databases excel in handling interconnected data, making them ideal for social networks, recommendation engines, and fraud detection systems. They provide quick traversal of relationships between data elements, enabling complex queries across vast networks. Neo4J recognized for its ease of use, scalability, and expressive query language, and Amazon Neptune, offering high performance and compatibility with graph query languages, are notable examples in this category.

5. Multi-model NoSQL: This type supports multiple data models within a single database system. It offers the flexibility to use different models, such as document, graph, or key-value, in a unified environment. Multi-model databases are advantageous when diverse data models are required without complex migrations or integration efforts. OrientDB, known for its ability to handle documents, graphs, object-oriented, and key-value models within one database engine, and ArangoDB, offering a unified approach to document, graph, and key-value models with a single query language, are prominent examples. They suit scenarios demanding versatile data modelling, providing a unified solution for varied data needs.

To take a look at some of the key differences between NoSQL DBMS and RDBMS let’s compare MongoDB and Postgres.

1. Data Model and Flexibility: MongoDB is a document-oriented NoSQL database, that stores data in flexible, JSON-like documents. It's schema-less, allowing each document to have different structures, ideal for evolving or unstructured data. Postgres, on the other hand, is a relational SQL database that enforces a predefined schema, ensuring data consistency but requiring predefined tables and relationships.

2. Query Language and Transactions: MongoDB uses a rich query language, including aggregation pipelines and native JSON query support, optimising document retrieval. However, complex transactions can be challenging due to its eventual consistency model. Postgres, with its SQL-based querying, excels in handling complex transactions and joins, ensuring ACID compliance for robust transactional support.

3. Performance and Scalability: MongoDB's horizontal scaling capability enables it to handle large data volumes across distributed systems, providing high-speed read/write operations. Its sharding architecture supports scalability. Postgres traditionally works well with smaller to medium-sized datasets, delivering strong performance for complex queries, but may require additional effort to scale horizontally.

4. Use Cases and Application: MongoDB is well-suited for applications demanding flexibility, rapid development, and scalability, such as content management systems, real-time analytics, and IoT applications. Postgres is often preferred for applications requiring strong data integrity, complex relationships, and ACID compliance, like financial systems, CRM, and data warehousing.

References

Ashtari, H. (2022, October 18). What Is NoSQL? Features, Types, and Examples. Spiceworks. https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-nosql/

Hiteshreddy2181. (2023, May 14). Geeks for Geeks. Types of NoSQL Databases. https://www.geeksforgeeks.org/types-of-nosql-databases/

IBM Technology. (2021, August 24). What is MongoDB? Youtube. https://www.youtube.com/watch?v=VOLeKvNz-Zo

IBM Technology. (2021, September 23). SQL vs. NoSQL: What's the difference? Youtube. https://www.youtube.com/watch?v=Q5aTUc7c4jg&list=PLOspHqNVtKAAXDobTc9kBWwnfgzNV2k_a

IBM Technology. (2022, June 29). Relational vs. Non-Relational Databases. Youtube. https://www.youtube.com/watch?v=E9AgJnsEvG4

MongoDB (nd). What is NoSQL? https://www.mongodb.com/nosql-explained

Ravoof, S. (2023, July 2023). MongoDB vs PostgreSQL: 15 Critical Differences. Kinsta. https://kinsta.com/blog/mongodb-vs-postgresql/

Taylor, D. (2023, October 19). Guru99. NoSQL Tutorial: What is, Types of NoSQL Databases & Example. https://www.guru99.com/nosql-tutorial.html