Instructions

For each answer, please include your answer as text, and any screenshot(s) which demonstrate your answer was executed. Most importantly, make sure to include evidence your answer is correct. This will most likely be a screenshot. If you had issues, problems, or had to make assumptions include them in your answer.

Your Answers:

1. Design your own scenario for which a Cassandra table would be a good solution. Make sure to explain the scenario and the specific characteristics of the scenario which would make Cassandra a good fit. Make sure to follow a query first approach and justify how the partition and cluster keys should be setup.

   IoT application on an apple watch or other similar device that tracks heart rate, blood sugar, body temperature, etc. This is good because it requires a large volume of writes, its time series, it does not require normalized tables, and reads can be subject specific. The partition key would be the type of measurement and the cluster key would be the timestamp.

2. Create your Cassandra table in CQL based on your scenario from the previous exercise. You should define the columns and data types to suit your scenario in addition to configuring the partition and cluster keys.

   first create a new keyspace
   > create keyspace device with replication = { 'class' : 'SimpleStrategy', 'replication_factor' : 3 };

```
cqlsh> create keyspace device with
   ... replication = {'class' : 'SimpleStrategy',
   ... 'replication_factor' : 3}
   ... ;
cqlsh> describe keyspaces;

system_schema  system_auth  system  system_distributed  device  system_traces

cqlsh>
```

   create the Cassandra table
   > use device
   > create table health_measurements (health_measure_type text, health_measure_time timestamp, health_measure_value float, primary key (health_measure_type, health_measure_time));

```
cqlsh> use device
   ... ;
cqlsh:device> create table health_measurements (
          ... health_measure_type text,
          ... health_measure_time timestamp,
          ... health_measure_value float,
          ... primary key (health_measure_type, health_measure_time)
          ... );
cqlsh:device>
```

3. Write CQL statements to add data to your table. Add at least 9 records consisting of 3 different partition and cluster keys

   add 9 records to the table
   > Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('heart_rate', '2020-05-01 08:00', 120);
   > Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('heart_rate', '2020-05-01 09:00', 130);
   > Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('heart_rate', '2020-05-01 09:00', 125);
   > Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('blood_sugar', '2020-05-01 08:00', 90);
   > Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('blood_sugar', '2020-05-01 09:00', 100);
   > Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('blood_sugar', '2020-05-01 10:00', 110);
   > Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('body_temp', '2020-05-01 08:00', 99.7);
   > Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('body_temp', '2020-05-01 09:00', 100);
   > Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('body_temp', '2020-05-01 10:00', 99.2);

```
cqlsh:device> Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('
heart_rate', '2020-05-01 09:00', 130);
cqlsh:device> Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('
heart_rate', '2020-05-01 10:00', 125);
cqlsh:device> Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('
blood_sugar', '2020-05-01 08:00', 90);
cqlsh:device> Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('
blood_sugar', '2020-05-01 09:00', 100);
cqlsh:device> Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('
blood_sugar', '2020-05-01 10:00', 110);
cqlsh:device> Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('
body_temp', '2020-05-01 08:00', 99.7);
cqlsh:device> Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('
body_temp', '2020-05-01 09:00', 100);
cqlsh:device> Insert into health_measurements (health_measure_type, health_measure_time, health_measure_value) values ('
body_temp', '2020-05-01 10:00', 99.2);
cqlsh:device>
```

```
cqlsh:device> select * from health_measurements;

 health_measure_type | health_measure_time             | health_measure_value
---------------------+---------------------------------+----------------------
          heart_rate | 2020-05-01 08:00:00.000000+0000 |                  120
          heart_rate | 2020-05-01 09:00:00.000000+0000 |                  130
          heart_rate | 2020-05-01 10:00:00.000000+0000 |                  125
           body_temp | 2020-05-01 08:00:00.000000+0000 |                 99.7
           body_temp | 2020-05-01 09:00:00.000000+0000 |                  100
           body_temp | 2020-05-01 10:00:00.000000+0000 |                 99.2
         blood_sugar | 2020-05-01 08:00:00.000000+0000 |                   90
         blood_sugar | 2020-05-01 09:00:00.000000+0000 |                  100
         blood_sugar | 2020-05-01 10:00:00.000000+0000 |                  110

(9 rows)
cqlsh:device>
```

I added in an extra column after the fact

```
cqlsh:device> select * from health_measurements;

 health_measure_type | health_measure_time             | health_measure_value | priority
---------------------+---------------------------------+----------------------+----------
          heart_rate | 2020-05-01 08:00:00.000000+0000 |                  120 |        1
          heart_rate | 2020-05-01 09:00:00.000000+0000 |                  130 |        1
          heart_rate | 2020-05-01 10:00:00.000000+0000 |                  125 |        1
           body_temp | 2020-05-01 08:00:00.000000+0000 |                 99.7 |        3
           body_temp | 2020-05-01 09:00:00.000000+0000 |                  100 |        3
           body_temp | 2020-05-01 10:00:00.000000+0000 |                 99.2 |        3
         blood_sugar | 2020-05-01 08:00:00.000000+0000 |                   90 |        2
         blood_sugar | 2020-05-01 09:00:00.000000+0000 |                  100 |        2
         blood_sugar | 2020-05-01 10:00:00.000000+0000 |                  110 |        2

(9 rows)
cqlsh:device>
```

4. Write a CQL statement to create an index or materialized view on your table so that you can set a different partition key to prevent ALLOW FILTERING. Then write a CQL SELECT statement to demonstrate it works as designed.

   show that the query doesn't work without allow filtering clause
   > select * from health_measurements where priority = '1';

```
cqlsh:device> select * from health_measurements where priority = '1';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data
 filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpred
ictability, use ALLOW FILTERING"
cqlsh:device>
```

create a materialized view on priority

```
cqlsh:device> create materialized view health_measurements_by_priority
          ... as select * from health_measurements
          ... where priority is not null and health_measure_type is not null and health_measure_time is not null
          ... primary key (priority, health_measure_type, health_measure_time);
cqlsh:device>
```

show that it is there

```
cqlsh:device> describe health_measurements;

CREATE TABLE device.health_measurements (
    health_measure_type text,
    health_measure_time timestamp,
    health_measure_value float,
    priority text,
    PRIMARY KEY (health_measure_type, health_measure_time)
) WITH CLUSTERING ORDER BY (health_measure_time ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32',
 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';

CREATE MATERIALIZED VIEW device.health_measurements_by_priority AS
    SELECT *
    FROM device.health_measurements
    WHERE priority IS NOT NULL AND health_measure_type IS NOT NULL AND health_measure_time IS NOT NULL
    PRIMARY KEY (priority, health_measure_type, health_measure_time)
    WITH CLUSTERING ORDER BY (health_measure_type ASC, health_measure_time ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32',
 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
```

show that the query works without allow filtering clause
> select * from health_measurements_by_priority where priority = ‘1’;

```
cqlsh:device> select * from health_measurements_by_priority where priority = '1';

 priority | health_measure_type | health_measure_time             | health_measure_value
----------+---------------------+---------------------------------+----------------------
        1 |          heart_rate | 2020-05-01 08:00:00.000000+0000 |                  120
        1 |          heart_rate | 2020-05-01 09:00:00.000000+0000 |                  130
        1 |          heart_rate | 2020-05-01 10:00:00.000000+0000 |                  125

(3 rows)
cqlsh:device>
```

5. Write a CQL statement to create an index or materialized view on your table so that you can set a different cluster key to prevent ALLOW FILTERING. Then write a CQL SELECT statement to demonstrate it works as designed.

   show that the query doesn't work without allow filtering clause
   > select * from health_measurements where health_measurement_time > '2020-05-01 08:00';

```
cqlsh:device> select * from health_measurements where health_measure_time > '2020-05-01 08:00';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data
 filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpred
ictability, use ALLOW FILTERING"
cqlsh:device>
```

   create a secondary index on priority
   > create index ix_health_measure_priority on health_measurements (priority);

```
cqlsh:device> create materialized view health_measurements_by_priority
          ... as select * from health_measurements
          ... where priority is not null and health_measure_type is not null and health_measure_time is not null
          ... primary key (priority, health_measure_type, health_measure_time);
cqlsh:device>
```

   show that it is there

```
CREATE TABLE device.health_measurements (
    health_measure_type text,
    health_measure_time timestamp,
    health_measure_value float,
    priority text,
    PRIMARY KEY (health_measure_type, health_measure_time)
) WITH CLUSTERING ORDER BY (health_measure_time ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'ma
 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Comp
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
CREATE INDEX ix_health_measure_priority ON device.health_measurements (priority);
```

   show that the query works without allow filtering clause
   > select * from health_measurements where health_measurement_type = 'blood_sugar' and health_measurement_time > '2020-05-01 08:00';

```
cqlsh:device> select * from health_measurements where health_measure_type = 'blood_sugar' and health_measure_time > '202
0-05-01 08:00';

 health_measure_type | health_measure_time             | health_measure_value | priority
---------------------+---------------------------------+----------------------+----------
        blood_sugar | 2020-05-01 09:00:00.000000+0000 |                  100 |        2
        blood_sugar | 2020-05-01 10:00:00.000000+0000 |                  110 |        2

(2 rows)
cqlsh:device>
```