

MIT 212 Advanced Database Design and Development

Smart Streetlight Management System

Submitted By:

Lushant Shrestha 988241

Aashish Adhikari 987986

Hasibur Rahman 988065

HM Adnan Zami 987963

Lecturer: Dr. Mohamad Sadeghi

Session: Semester 1, 2026

Submission Date: 28 May 2026

Table of Contents

1. Sustainable Development Context 3
In our case study related to SDG 7 and SDG 11 3
Example 3
2. NoSQL Database Paradigms 4
3. MongoDB and Redis Evaluation 5
3.1 MongoDB as the Document-Oriented Implementation 5
3.2 Redis as the Key-Value and Operational Implementation 6
3.3 Comparative Evaluation of MongoDB and Redis 7
3.4 Critical Judgement 8
4. Data Modelling for Smart and Sustainable Cities 9
4.1 MongoDB Document Model 9
4.2 MongoDB Maintenance Record Model 10
4.3 Redis Key-Value and Data-Structure Model 11
4.4 Redis Maintenance and Energy Model 11
4.5 Comparison of MongoDB and Redis Data Modelling 12
4.6 Data Quality, Consistency and Sustainability Value 13
5. Applications and Industry Use Cases 14
5.1 Practical Application of the Smart Streetlight System 14
5.2 MongoDB for Sustainability Analysis and Planning 14
5.3 Redis for Rapid Operations and Maintenance Response 15
5.4 Smart City Scenario 15
5.5 Sustainability and Sustainable Development Goals 16
SDG 7: Affordable and Clean Energy 16
SDG 11: Sustainable Cities and Communities 16
5.6 Industry Applications 16
5.7 Critical Evaluation of Sustainability Value 17
6. MongoDB and Redis Comparison 17
6.1 Comparison of Update Time 18
6.2 Comparison of Read 19
6.3 Comparison of Insert Time 20
In connection with our case study 20
References 22

1. Sustainable Development Context

The United Nations Sustainable Development Goals (SDGs) is a global framework that was introduced to help countries, organisations and communities move towards becoming more sustainable by 2030. The SDGs encompass economic, social and environmental goals such as addressing poverty, clean energy, climate action, sustainable cities, innovation and responsible consumption. There are 17 goals in the UN agenda, which are designed to solve the global problems with measurable targets and through the cooperation of the countries.

There are strong linkages between smart and sustainable cities and several SDGs. SDG 11: Sustainable Cities and Communities is the most relevant SDG in terms of making cities inclusive, safe, resilient and sustainable. Smart city systems can also contribute to SDG 7: Affordable and Clean Energy, SDG 9: Industry, Innovation and Infrastructure, SDG 12: Responsible Consumption and Production, and SDG 13: Climate Action. For instance, smart energy monitoring systems can be used to cut waste and intelligent transport and infrastructure monitoring systems support SDG 11 and SDG 9.

In our case study related to SDG 7 and SDG 11

Data-driven systems are also crucial in Sustainable Urban Development as they can gather, analyse and process real-time data. Sensors and IoT devices, mobile applications and digital platforms are used in smart cities to track traffic, waste collection, water usage, air quality and energy consumption. The information can be used to assist city planners in making decisions, prevent wasting of resources and enhancing service quality.

Smart Streetlight Management System is an urban sustainable infrastructure system project that aims to optimize the performance of urban street lighting systems. This system provides city infrastructure managers with a way to track and control the lights on the street, including data such as location, type, power rating, operational history, maintenance history, and energy usage. The principal goal of this project is to lower the amount of wasted energy, promote timely maintenance, and enhance public safety with improved lighting management. This project is aligned with Sustainable Development Goal (SDG) 11 – Sustainable Cities and Communities and SDG 7 – Affordable and Clean Energy as it encourages smarter energy use and an environmentally friendly city operation. The system will be created in two separate versions with both document-oriented (MongoDB) and key-value (Redis) NoSQL database technologies, and the performance, memory consumption, and data management efficiency of both different NoSQL database approaches will be compared.

Example

Smart Streetlight Management System is a high-level smart city solution, which is used for the improvement of efficiency, sustainability and reliability in the management of public street lights. The system allows city infrastructure managers to monitor and control streetlights and record key information like location, street light type, power rating, operation zone, operational status, maintenance history, and energy consumption. It aims for the main goal of minimizing any energy wastage, detecting and maintaining faults on time, decreasing the operational costs and increasing the public safety by efficient lighting management. The project contributes directly, and in various ways, to Sustainable Development Goal (SDG) 11 – Sustainable Cities and Communities, and SDG 7 – Affordable and Clean Energy, by advocating for sustainable urban infrastructure and smarter energy usage. For comparison of NoSQL database technologies, the system will be implemented independently using two open source

projects, MongoDB and Redis, to enable you to test performance, speed, memory usage and data management in a real-world smart infrastructure application.

The importance of databases is highlighted in this context as these hold and process the vast amount of data created by smart city systems. Older systems might not be able to handle large, diverse, and real-time data. Thus, advanced database systems like MongoDB and Redis will become valuable tools for smart city applications, enhancing scalability, performance, and adaptability.

2. NoSQL Database Paradigms

No SQL databases are non-relational database solutions that can be used to store and manage data that is not necessarily contained in tables. NoSQL databases have more flexible data models than relational databases, which rely on rows, columns, and pre-defined schemas. NoSQL databases have more flexible data models than relational databases, which are based on rows, columns, and pre-defined schemas. This makes them suitable for modern applications that generate a lot of structured, semi-structured and unstructured data. In the Microsoft Azure perspective, NoSQL databases are non-relational and are ideal for data that doesn't have a fixed structure in tables and offer flexibility, scalability and performance.

NoSQL databases have flexible schemas, multiple copies across the network so they can be made available everywhere, scale horizontally, and perform a large number of read/write operations per second. Schema flexibility enables applications to store various types of data without having to redesign the database significantly. Horizontal scalability is essential in large-scale systems like smart city platforms, as it enables distributing data among multiple servers. High availability means that services are available even if a part fails.

However, traditional relational databases are not suitable for large-scale and real-time urban systems. Smart cities continuously gather data from sensors, energy meters, transportation systems, and public services, among other sources. This can come as fast and in various formats. Well-structured data is well-suited to a relational database, but as the need to process millions of real-time events arises, scaling can become a challenge. Relational databases also have the disadvantage of needing pre-specified schemas, which might not be appropriate if the form of the sensors or the application needs are subject to rapid change.

For sustainability applications, flexible and complex data records that can be stored in document-oriented databases like MongoDB are relevant. MongoDB uses a flexible schema where the structure of a document can change as application design evolves, and stores data in JSON-like documents (Marat Nuriev et al., 2024)

In the case of a smart city platform, for instance, data on location, time, temperature, energy consumption, pollution and the status of the device can be maintained in the same document. This is important as various sensors can generate different fields.

Another type of database used in sustainability applications is a key-value database like Redis, which offers extremely fast data access. Data is stored in Redis with keys and values and is frequently used in situations that require low-latency data retrieval. As noted in the Redis documentation, Redis allows for

storing and retrieving data with keys and can provide fast access patterns. For smart city systems, Redis can serve as temporary sensor reading data, traffic signal states, user sessions, and alerts.

Smart Streetlight Management System is a case study on managing public street lighting efficiently in a smart and sustainable way to develop an urban infrastructure solution. The system can be used to aid city infrastructure managers in monitoring the operation of streetlights, managing maintenance activities, tracking energy consumption, and quickly locating defective or high-energy-consuming streetlights. It is primarily used to optimise energy use, save on maintenance and improve public safety by utilising superior streetlight management. This project closely aligns with Sustainable Development Goal 11 (Sustainable Cities and Communities) and SDG 7 (Affordable and Clean Energy), both of which promote the use of cleaner energy and better city management. Two NoSQL technologies will be compared, with the system developed with both technologies, and the performance, speed and data-handling capabilities assessed.

NoSQL databases are very applicable to smart and sustainable cities due to their speed, flexibility and scalability. The ability to store rich urban data comes in handy for MongoDB, and real-time access and responses for Redis. They can work together to create sustainable systems that need to store data over the long term and make decisions in real time.

3. MongoDB and Redis Evaluation

This project implements the Smart Streetlight Management System using two NoSQL database approaches: MongoDB and Redis. Both versions are designed around the same practical problem of managing streetlights, maintenance activities, operational status and energy-related information. However, the way each database stores and retrieves data is different. This difference is important because smart city systems need both reliable long-term records and fast access to operational information.

MongoDB represents the document-oriented approach. It is suitable for storing detailed streetlight and maintenance records that may later be searched, filtered or analysed. Redis represents the key-value and data-structure approach. It is suitable for retrieving current status quickly, supporting maintenance actions and storing energy-related operational data in a fast-access format. Therefore, this evaluation does not treat one database as automatically better than the other. Instead, it considers how each database supports different workload needs within the same smart streetlight context.

3.1 MongoDB as the Document-Oriented Implementation

MongoDB is a document-oriented NoSQL database that stores data in flexible BSON documents. This means that a real-world object, such as one streetlight, can be stored as one document containing several related fields. MongoDB is suitable for applications where records may change over time or where additional data may later be added. According to MongoDB documentation, its flexible document model allows the data structure to develop with the needs of the application while still supporting planned schema design and validation (MongoDB, n.d.-a; MongoDB, n.d.-b).

In the MongoDB implementation of the Smart Streetlight Management System, the database is named `Smart_Streetlight_System`. It uses two main collections: `Streetlights_info` and

Maintenance_records. The Streetlights_info collection stores the main streetlight asset information, including zoneID, streetlightID, installationDate, location, lightType, powerRating and lightStatus. The Maintenance_records collection stores maintenance-related information, including recordID, streetlightID, date, technicianName, description and maintenance status.

This design is appropriate because streetlight asset information and maintenance history serve different purposes. A streetlight record represents the physical asset and its current condition. A maintenance record represents an event or action related to that asset. In practice, one streetlight may require several maintenance records over its lifetime. Keeping these records separate makes the database easier to manage and supports longer-term maintenance analysis.

The MongoDB version also supports the main operations required by the system. It allows users to create, view, update and remove streetlight records. It can search for faulty streetlights by checking the lightStatus field, and it can identify lights with a powerRating above a selected threshold. The system also uses MongoDB aggregation to match records by zoneID and calculate the total connected power rating for a zone.

This analytical capability is one of MongoDB's main strengths in the project. For example, a council could use the stored data to identify zones with higher connected power load, compare LED and Halogen streetlights, or review maintenance patterns over time. From a sustainability perspective, this information can help support lighting upgrades and more efficient maintenance planning.

However, the current MongoDB implementation has an important limitation. It stores and analyses powerRating in watts, rather than recording full energy consumption in kilowatt-hours over time. Power rating can help identify potentially high-power assets, but it does not by itself show actual daily energy use. To calculate energy usage in kWh, the system would also need operating hours or sensor-based readings. This distinction is important because sustainability reporting should be based on technically accurate measurements.

MongoDB also requires careful data control. Its flexible structure is useful, but inconsistent values can reduce the quality of searching and reporting. The use of input validation in the project helps address this by checking dates, numeric values, menu choices and empty inputs before records are stored. Overall, MongoDB is well suited to the detailed record management and analytical side of the Smart Streetlight Management System.

3.2 Redis as the Key-Value and Operational Implementation

Redis is an in-memory NoSQL database that stores data through keys and supporting data structures such as hashes, sets and sorted sets. Redis hashes are suitable for representing objects through field-value pairs, while sets and sorted sets are useful for grouping items and maintaining ordered or time-based relationships (Redis, n.d.-a; Redis, n.d.-b).

In the Redis implementation of the Smart Streetlight Management System, streetlight data is stored using key patterns such as streetlight:{id}. This key stores the main streetlight information as a hash, including fields such as zone, installation date, location, light type, power rating and light status. The implementation also uses a separate key pattern, streetlight:{id}:status, to store current operational values such as the current status, the last status change and accumulated power-consumption information.

Redis also organises streetlights using supporting keys. For example, `zone:{zone}:lights` is used to store the streetlights that belong to a particular zone, while `streetlights:all` stores the list of active streetlights. This makes it possible to retrieve operational data quickly without performing the same type of document search used in MongoDB.

The Redis implementation also supports maintenance records. Each maintenance entry is stored using a key pattern such as `maintenance:{id}`. In addition, the key `streetlight:{id}:maintenance` uses a sorted set to maintain a time-based relationship between a streetlight and its maintenance records. This is a useful design because the system can retrieve a maintenance timeline for a specific streetlight while still keeping individual maintenance records separate.

A particularly important feature in the Redis implementation is its energy-related model. The code includes key patterns such as `energy:daily:light:{id}` and `energy:daily:zone:{zone}`. These keys allow the system to record daily consumption values for an individual streetlight and update the corresponding zone total. This means the Redis implementation goes beyond simple power-rating checks by including a structure for storing daily energy usage values in kWh.

Redis is therefore useful for the operational side of the system. When a streetlight changes status, requires maintenance, or records new energy information, the system can update the relevant Redis keys quickly. This is suitable for current-status retrieval, quick maintenance action, operational dashboards and alert-style monitoring.

However, Redis also introduces design challenges. Since related information is spread across multiple keys, the application must update those keys consistently. For example, if a streetlight is removed or its status changes, the system must update the main streetlight hash, status information and any relevant zone or maintenance structures. If one related key is missed, the system may return inconsistent information. Redis is also less natural than MongoDB for complex historical reporting unless the required access patterns are designed carefully in advance.

Overall, Redis is suitable for the fast operational and energy-recording parts of the Smart Streetlight Management System. It provides rapid access to current information, but it requires careful key management to maintain consistency.

3.3 Comparative Evaluation of MongoDB and Redis

MongoDB and Redis support the same Smart Streetlight Management System, but they do so through different data models. MongoDB stores records as documents in collections, while Redis stores information through hashes, sets, sorted sets and structured keys. This difference affects the way each database supports sustainability, maintenance operations and system performance.

Comparison Area	MongoDB Implementation	Redis Implementation
Database model	Document-oriented NoSQL database	In-memory key-value and data-structure database
Main data organisation	Collections and BSON-style documents	Hashes, sets, sorted sets and structured keys
Streetlight storage	`Streetlights_info` collection	`streetlight:{id}` hash
Maintenance storage	`Maintenance_records` collection	`maintenance:{id}` hash and maintenance timeline sorted set

Status monitoring	Searches documents by `lightStatus`	Stores current operational status in separate status keys
Power and energy analysis	Uses `powerRating` and zone aggregation	Supports `powerRating` checks and daily energy keys
Strongest practical use	Detailed records, historical storage and sustainability analysis	Fast operational access, maintenance response and daily energy updates
Query style	Filtering and aggregation are more natural	Access is fast, but depends on careful key design
Main limitation	Less suitable than Redis for repeated live access operations	More complex to maintain consistency across related keys

The comparison shows that MongoDB is stronger for detailed record storage and analytical work. Its collection structure makes it easier to keep streetlight assets and maintenance records organised, while its aggregation capability supports zone-based power-load analysis. This is useful for long-term sustainability planning because decision-makers need historical data to identify inefficient assets and repeated maintenance problems.

Redis is stronger for operational speed and quick actions. Its key-based design supports direct access to a streetlight, its current status, related maintenance records and daily energy values. This is useful where a system needs frequent status updates or fast response to faults. However, this advantage depends on careful data management because related Redis keys must remain consistent when records are changed.

There is also an important implementation difference between the two versions. The MongoDB version currently analyses `powerRating` and zone-level connected power load. The Redis version includes additional daily energy structures that can store consumption values in kWh for individual streetlights and zones. This difference should be acknowledged because a fair empirical comparison requires the same operations and data workload to be tested in both systems. From a critical evaluation perspective, the difference suggests that the two implementations are functionally similar in their core streetlight and maintenance operations, but they are not fully identical in their current energy-tracking detail.

3.4 Critical Judgement

The project demonstrates that database suitability depends on the purpose of the workload. MongoDB is not simply a slower alternative to Redis, and Redis is not automatically the best choice because it performs fast operations. MongoDB provides a stronger structure for detailed streetlight records, maintenance history and analytical reporting. Redis provides a stronger structure for quick operational retrieval, current status handling and daily energy updates.

For the assessment, the two databases are implemented independently so that their performance can be compared fairly. In a real smart city deployment, however, the two technologies could also work together. MongoDB could store the long-term streetlight and maintenance history required for reporting and sustainability planning, while Redis could support live status monitoring, maintenance alerts and frequently updated operational values. This is important because sustainable urban infrastructure requires both quick response to current problems and reliable historical evidence for future decisions.

Therefore, the most appropriate conclusion is not that one database replaces the other. MongoDB and Redis each provide value for the Smart Streetlight Management System, but their value appears in different parts of the workload. MongoDB supports memory, history and analysis; Redis supports speed, current state and operational response.

4. Data Modelling for Smart and Sustainable Cities

Data modelling is a central part of the Smart Streetlight Management System because it determines how streetlight assets, maintenance activities, operational status and energy-related data are stored and retrieved. In a smart city system, the database model must support more than simple record keeping. It should also help identify faults, organise maintenance work, review power use and support sustainability decisions.

This project implements the same streetlight management problem using two NoSQL models. MongoDB stores data as documents in collections, while Redis stores data through structured keys and data structures such as hashes, sets and sorted sets. Both models support streetlight management, but they organise the information differently. From a database design perspective, MongoDB is more natural for detailed records and analysis, while Redis is more suitable for rapid access to current operational information.

4.1 MongoDB Document Model

The MongoDB implementation uses a database named `Smart_Streetlight_System` with two main collections:

Collection	Purpose
`Streetlights_info`	Stores the main information for each streetlight asset
`Maintenance_records`	Stores fault descriptions, technician details, repair dates and maintenance outcomes

The `Streetlights_info` collection stores one document for each streetlight. Based on the implemented Python code, a simplified document is shown below:

```
{
  "zoneID": "z1",
  "streetlightID": "z1sl101",
  "installationDate": "18-05-2026",
  "location": "Main Street",
  "lightType": "LED",
  "powerRating": 45.0,
  "lightStatus": "ON"
}
```

This model is practical because the main details of one physical asset are kept together in one document. The `streetlightID` field allows the system to search for, update or remove a particular streetlight. The `zoneID` field supports analysis by area. The `lightStatus` field supports faulty-light identification, while the `powerRating` field supports checks for streetlights with higher connected power load.

MongoDB suits this kind of data because the document structure can be extended later. For example, a future version could add operating hours, motion sensor readings, brightness levels, installation cost or carbon reduction estimates without redesigning the entire database. This flexibility is useful in smart city applications because the type of data collected often becomes more detailed over time.

The MongoDB implementation also uses aggregation to calculate the total `powerRating` of streetlights within a zone. This supports an early form of sustainability analysis because it can show which zones

have higher connected lighting load. However, this value represents watts, not actual energy consumed in kWh over time. This limitation needs to be recognised so that the analysis remains technically accurate.

For improved query performance, suitable indexes could be added on common search fields such as:

Recommended Index Field	Reason
`streetlightID`	Speeds up lookup, update and deletion of one streetlight
`zoneID`	Supports zone-based retrieval and aggregation
`lightStatus`	Helps quickly find faulty lights
`powerRating`	Supports high-power streetlight searches

These indexes would be particularly useful if the number of streetlights increases across a larger urban area.

4.2 MongoDB Maintenance Record Model

The second MongoDB collection is `Maintenance_records`. A simplified document based on the implemented code is shown below:

```
{
  "recordID": "A92kL",
  "streetlightID": "z1sl101",
  "date": "20-05-2026",
  "technicianName": "Technician A",
  "description": "Bulb not working",
  "lightStatus": "pending"
}
```

Keeping maintenance records in a separate collection is a suitable design choice. One streetlight may experience several faults, inspections or repairs during its operational life. If every maintenance event was permanently embedded inside the main streetlight document, the document could grow unnecessarily and become harder to manage.

The `streetlightID` field provides the logical relationship between a streetlight and its maintenance history. This allows the asset record to remain focused on the streetlight itself, while the maintenance collection records the activities that occur over time. In practice, this supports questions such as:

- Which streetlights have repeated faults?
- Which repairs are still pending?
- Which assets have been decommissioned?
- Which zones require more maintenance attention?

This is important for sustainability because repeated repairs may suggest that replacement with a more efficient or reliable lighting asset is a better long-term decision.

4.3 Redis Key-Value and Data-Structure Model

The Redis implementation models the same system through structured keys rather than MongoDB collections. Redis uses hashes to store detailed records, sets to organise membership, sorted sets to represent time-based relationships, and additional hashes to record daily energy information.

The actual Redis key patterns used in the implementation are summarised below:

Redis Key Pattern	Data Structure	Purpose
`streetlight:{id}`	Hash	Stores the main streetlight record
`streetlight:{id}:status`	Hash	Stores current status, latest status change and accumulated consumption
`zone:{zone}:lights`	Set	Stores streetlight IDs belonging to one zone
`streetlights:all`	Set	Stores active streetlight IDs
`maintenance:{id}`	Hash	Stores one maintenance record
`streetlight:{id}:maintenance`	Sorted set	Stores a time-based maintenance timeline for one streetlight
`energy:daily:light:{id}`	Hash	Stores daily energy values for one streetlight
`energy:daily:zone:{zone}`	Hash	Stores daily energy values aggregated by zone

A streetlight record in Redis is stored as a hash. The structure can be understood as follows:

```
streetlight:z1sl101
streetlightID = z1sl101
zoneID = z1
location = Main Street
lightType = LED
powerRating = 45.0
lightStatus = on
is_active = true
```

This design supports quick access to one streetlight because the system can retrieve the hash directly from its key. Redis also creates a separate status hash:

```
streetlight:z1sl101:status
```

This status structure stores current operational information such as:

```
current_status = on
last_status_change = timestamp
power_consumption = 0
brightness_level = 100
```

This is a meaningful design choice for live operations. The main streetlight hash stores the asset record, while the status key keeps values that may change more frequently. In a real smart lighting environment, current status and brightness can change regularly, so separating them supports faster operational access.

4.4 Redis Maintenance and Energy Model

Redis also models maintenance information using a hash for each maintenance record:

```
maintenance:A92kL
```

The maintenance hash contains fields such as record ID, streetlight ID, maintenance date, technician name, description and maintenance status. In addition, the implementation uses a sorted set:

```
streetlight:z1sl101:maintenance
```

This sorted set stores maintenance record IDs using timestamps as scores. As a result, the system can maintain a chronological repair history for each streetlight. This is a strong Redis design choice because sorted sets are suitable when items need to be ordered by time.

The Redis version also includes a more detailed energy-related model than the current MongoDB implementation. It records daily energy usage using:

```
energy:daily:light:{id}
energy:daily:zone:{zone}
```

When energy consumption is recorded for a streetlight, the system updates both the daily total for that streetlight and the daily total for its zone. It also updates the streetlight status record by increasing its accumulated `power_consumption` value.

This model is valuable for sustainability analysis because it moves beyond simply storing a streetlight's rated wattage. It provides a structure for recording actual daily consumption values in kWh. This would allow the system to answer more useful sustainability questions, such as:

- Which streetlight consumed the most energy during a particular day?
- Which zone recorded the highest daily energy total?
- Are particular lights consuming more energy than expected?
- Should older or inefficient lights be prioritised for replacement?

Redis is therefore not only used for rapid status operations in this implementation. It also includes a structure for short-term operational energy monitoring.

4.5 Comparison of MongoDB and Redis Data Modelling

The two implementations model the same streetlight system through different structures.

Data Modelling Area	MongoDB Model	Redis Model
Main streetlight data	One document in `Streetlights_info`	Hash stored as `streetlight:{id}`
Maintenance data	Documents in `Maintenance_records`	Hash stored as `maintenance:{id}`
Link between asset and maintenance	Shared `streetlightID` field	Sorted set `streetlight:{id}:maintenance`
Zone grouping	Query and aggregation using `zoneID`	Set `zone:{zone}:lights`
Faulty-light lookup	Query where `lightStatus = FAULTY`	Search/status structures for rapid lookup
High-power check	Query on `powerRating`	Iteration/check using stored hash values
Zone power calculation	MongoDB aggregation pipeline	Summation across IDs stored in zone set

Daily energy model	Not explicitly implemented in current version	Implemented using daily light and zone energy keys
Main modelling strength	Detailed records and analysis	Fast current-state and operational access

MongoDB provides a cleaner structure for detailed records because the information is contained within documents that can be queried and aggregated directly. Redis offers faster direct access, but the information is spread across several related keys. For example, updating a Redis streetlight may require updating the main hash, the status hash, zone membership and possibly energy or maintenance structures. This means Redis can be fast, but it requires more careful application logic to avoid inconsistent data.

A further issue is feature parity. The assignment requires the MongoDB and Redis versions to support the same system functionality so that their performance can be compared fairly. In the current code, both implementations support the common core functions of streetlight management, maintenance records, faulty-light detection, high-power checks and zone-based power analysis. However, the Redis implementation also includes explicit daily energy recording in kWh, while the MongoDB version currently focuses on `powerRating` in watts. This should be acknowledged in the report because any performance comparison should use equivalent operations and equivalent datasets.

4.6 Data Quality, Consistency and Sustainability Value

Both data models depend on accurate input. The project uses validation functions to check dates, positive numeric values, menu selections and empty inputs before information is recorded. This improves reliability because incorrect information would reduce the usefulness of fault reports, maintenance history and sustainability analysis.

For MongoDB, consistency depends on storing reliable document fields and values. For Redis, consistency also depends on updating related keys correctly. For example, when a streetlight becomes decommissioned, the Redis implementation must update the streetlight status and ensure the asset is no longer treated as active in the relevant key structures.

The sustainability value of the data model is shown below:

Data Item	How It Is Used	Sustainability Value
`powerRating`	Identifies higher-rated streetlights	Supports early energy-efficiency review
Daily kWh data in Redis	Records actual energy usage over time	Supports more accurate consumption monitoring
`lightType`	Distinguishes LED and Halogen lights	Supports upgrade and replacement planning
`zoneID`	Groups lights by geographical area	Supports area-based analysis
`lightStatus`	Identifies faulty or decommissioned lights	Supports timely repair and public safety
Maintenance history	Records repeated faults and repairs	Supports preventive maintenance decisions

Overall, the MongoDB and Redis data models support different but related aspects of the Smart Streetlight Management System. MongoDB provides a clear structure for detailed asset and maintenance records, which is useful for historical analysis and planning. Redis provides a structured fast-access

model for current status, maintenance timelines and daily energy data, which is useful for operational monitoring. Together, the two implementations show how NoSQL database design can influence both system performance and sustainability outcomes.

5. Applications and Industry Use Cases

The Smart Streetlight Management System is a practical example of how database technologies can support sustainable urban infrastructure. Streetlights are important public assets because they affect road safety, pedestrian movement, energy consumption and maintenance costs. When streetlight data is organised properly, city managers can identify faults more quickly, analyse lighting performance and make better decisions about repair or replacement.

In this project, MongoDB and Redis support the same smart streetlight problem through different database approaches. MongoDB provides a document-based structure for detailed streetlight records, maintenance history and analytical reporting. Redis provides a key-value and data-structure model that supports quick access to current information, maintenance-related actions and daily energy values. This makes the system relevant not only as a database project, but also as a practical smart city solution.

5.1 Practical Application of the Smart Streetlight System

The system is designed for users such as city infrastructure managers, maintenance officers or local councils. These users need accurate information about streetlight assets, operational status, faults, maintenance activities and power-related information.

The core functions supported by the project are shown below:

System Function	Practical Purpose
Register streetlights	Adds a new lighting asset with location, zone, lamp type, status and power rating
View streetlight records	Allows managers to review current streetlight information
Update streetlight details	Records changes in status, location, lamp type or power rating
Remove streetlights	Removes assets that are no longer required or have been decommissioned
Identify faulty lights	Helps maintenance teams locate lights requiring attention
Identify high-power lights	Supports review of assets that may be less energy efficient
Add maintenance records	Stores fault descriptions, technician activity and repair status
Update maintenance records	Tracks whether a repair is pending, completed or decommissioned
Analyse by zone	Helps compare lighting load or energy-related data across areas

These functions are useful because public lighting networks can contain many assets spread across different locations. Without a database system, fault information and repair history may be difficult to track. The Smart Streetlight Management System provides a structured way to manage this information and support operational decision-making.

5.2 MongoDB for Sustainability Analysis and Planning

MongoDB is useful for the analytical and historical side of the application. Its document-oriented model stores detailed streetlight and maintenance records in a clear structure. This allows the system to retain

important information over time, including where a streetlight is located, what type of lamp it uses, whether it has experienced faults and what maintenance action has been completed.

The MongoDB implementation supports sustainability-related analysis through the `powerRating`, `lightType`, `zoneID` and `lightStatus` fields. For example, managers can identify streetlights with a higher power rating or compare the connected power load of different zones. They can also review whether a streetlight is an LED or Halogen model and consider whether older, less efficient assets should be replaced.

The maintenance records stored in MongoDB also contribute to sustainability planning. If the same streetlight requires repeated repair, replacing the asset may become more efficient than continuing to repair it. Similarly, if one zone records many faulty lights, that area may require a broader infrastructure inspection rather than individual repairs only.

It is important to state the current limitation accurately. The MongoDB version analyses connected power rating in watts, rather than storing actual daily energy consumption in kWh. Therefore, it supports early energy-efficiency review but not a complete energy consumption report. A future improvement could add operating hours or sensor readings so that actual energy use can be calculated and compared over time.

5.3 Redis for Rapid Operations and Maintenance Response

Redis is useful for the operational side of the application because its key-based model allows information to be accessed and updated quickly. In the Redis implementation, the system can store streetlight records, current status information, maintenance records, zone membership and daily energy-related values using structured keys.

This design is particularly useful when the system needs to respond quickly to operational events. For example, if a streetlight becomes faulty, the current status can be updated and retrieved rapidly so that maintenance action can begin sooner. Redis also maintains relationships between streetlights and their maintenance timeline, allowing recent or urgent activities to be accessed efficiently.

The Redis implementation includes daily energy key structures for individual streetlights and zones. This means Redis can support short-term operational monitoring of recorded kWh values. For example, a manager could use the system to observe daily energy values for a light or compare totals across zones. This is useful for identifying unusual consumption and prioritising further investigation.

However, Redis also requires careful consistency management. Since one streetlight may be represented across several related keys, changes must be updated correctly throughout the system. For example, if a streetlight is removed or its maintenance status changes, the relevant streetlight, zone, status, maintenance and energy structures must remain accurate. This means Redis provides speed and operational responsiveness, but it also requires disciplined application logic.

5.4 Smart City Scenario

A realistic use case can show how the database design supports smart city operations.

Consider a streetlight in a residential zone that becomes faulty during the evening. In the Redis implementation, the current status can be accessed or updated quickly so the system can support a rapid

maintenance response. A maintenance activity can then be recorded, including the streetlight ID, date, technician and fault description.

In the MongoDB implementation, the same type of streetlight and maintenance information can be retained as detailed documents. Over time, these records can help identify whether the same light has failed repeatedly or whether a particular zone experiences more faults than other zones. Managers can then decide whether repair, replacement or a wider lighting upgrade is the most suitable action.

This example shows the practical difference between the two database approaches. Redis supports quick operational response, while MongoDB supports detailed record keeping and longer-term analysis. In a real smart city architecture, these strengths could complement each other. In this assessment, they are implemented and evaluated as separate NoSQL versions of the same system.

5.5 Sustainability and Sustainable Development Goals

The Smart Streetlight Management System is directly relevant to **SDG 7: Affordable and Clean Energy** and **SDG 11: Sustainable Cities and Communities**.

SDG 7: Affordable and Clean Energy

Public streetlighting can contribute significantly to electricity use across urban areas. A system that identifies high-power assets and records energy-related information can support more efficient lighting decisions. In this project, MongoDB can support review of power rating and lamp type, while Redis includes daily energy-related structures that can support short-term consumption monitoring.

For example, if one area contains several high-power Halogen lights, the council may review whether these assets should be replaced with LED alternatives. If daily energy values show unusual consumption for a streetlight or zone, maintenance staff may investigate faults, inefficient operation or unsuitable equipment. In this way, the system supports energy efficiency rather than simply storing technical records.

SDG 11: Sustainable Cities and Communities

Streetlights are part of public infrastructure and directly affect safety, visibility and the reliability of urban services. Faulty streetlights can reduce pedestrian confidence and create safety concerns, particularly at night. A system that records faults and maintenance actions allows problems to be handled in a more organised way.

The project supports SDG 11 by enabling streetlight asset management, fault identification and maintenance tracking. Redis can support quicker operational response to current issues, while MongoDB can help analyse repeated faults and longer-term repair patterns. Together, these database approaches show how data-driven systems can improve the management of urban infrastructure.

5.6 Industry Applications

Although the project focuses on public streetlights, the same database design approach can be adapted to other asset-management environments.

Industry or Setting	Example Application

Local government	Manage public lighting assets, faults, repairs and upgrade planning
Roads and transport networks	Monitor lighting around roads, tunnels, bus stops and stations
University campuses	Track campus lighting, safety zones and maintenance work
Hospitals and public facilities	Manage emergency lighting and external infrastructure assets
Smart buildings	Monitor lighting status, maintenance and energy-related information
Industrial sites	Track operational equipment, faults and energy-efficiency concerns

The value of the database design is that it can be extended beyond one type of asset. Any organisation managing physical equipment can benefit from recording asset details, current status, maintenance history and energy-related measures.

5.7 Critical Evaluation of Sustainability Value

The system demonstrates a meaningful connection between database design and sustainability, but its contribution should not be overstated. A database alone does not reduce energy consumption or repair streetlights. Its value comes from providing accurate information that supports better decisions and faster actions.

MongoDB supports this by storing detailed and analysable records. Redis supports this by enabling rapid access to operational data and recorded daily energy values. If the data is incomplete, inconsistent or not acted upon, the sustainability benefit is limited. For example, identifying a high-power streetlight does not create energy savings unless the organisation uses that information to investigate, maintain or replace the asset.

Another important limitation is the difference between the two current implementations. The Redis version includes explicit daily energy recording structures, while the MongoDB version currently focuses mainly on power-rating analysis. For a fair database performance comparison, the same functions and equivalent data workloads should be tested in both systems. This is important because a database cannot be judged fairly if one version is completing more detailed operations than the other.

Overall, the Smart Streetlight Management System provides a practical foundation for sustainable infrastructure management. MongoDB is valuable for detailed historical records and sustainability analysis, while Redis is valuable for fast operational monitoring and maintenance response. Their different strengths demonstrate that suitable database design can help turn infrastructure data into more timely, efficient and sustainable urban decisions.

6. MongoDB and Redis Comparison

When evaluating database technologies for smart and sustainable city systems, it is a must to evaluate performance. The data will usually come in from the sensors, infrastructure, transport systems and energy networks, and the database must be capable of processing this data. The main KPIs are: Insertion time, Retrieval time, Update time, Resource usage, Availability and Scalability.

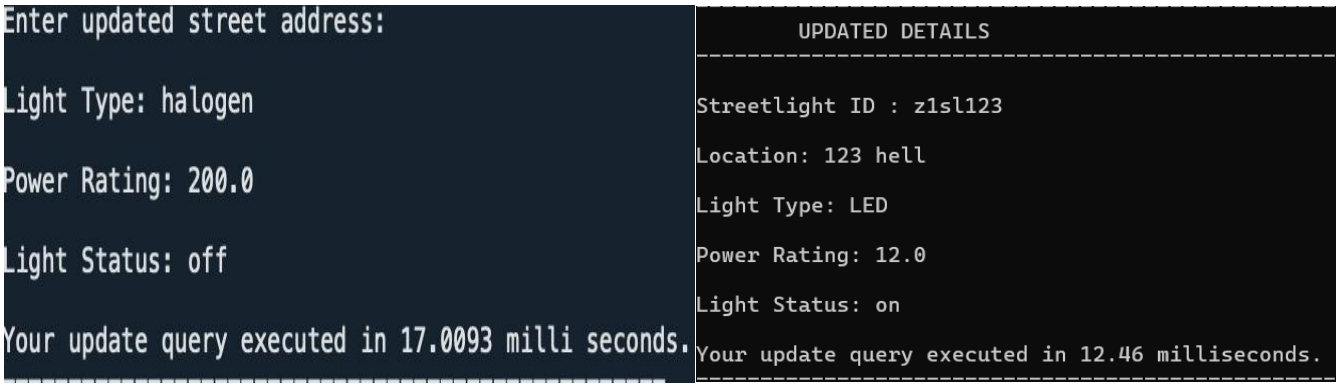
Insertion time: time that the database takes to store new data. In a smart city context the insertion performance is relevant since the sensors may continuously report on a regular basis, for instance every few seconds. For example, air quality sensors, smart electricity meters and traffic counters can log thousands of instances each minute. This insertion time is too slow and data will be delayed or lost. In this aspect, MongoDB may be advantageous because it allows for the storage of semi-structured

documents, which means the data stored within them is not required to be formatted in a specific manner, and it can handle large amounts of data. It needs to be stored in memory by default, meaning that the data would be stored in Redis and could be written to at extremely fast speeds.

The time it takes to store data, and retrieve it from storage. A critical application for dashboards, alerts and decision-making systems. Suppose there is a congestion system that a smart traffic system is tasked with showing the current congestion level, it has to get the data in a timely fashion. This is where Redis shines, as it is an in-memory data store and the best for real-time applications. Besides efficient retrieval with indexing and query, MongoDB is good for searching historical sustainability data.

The update time is the time required to update the latest records. An update of smart city systems can be generated by changes in the maintenance log, the status of sensors or the information on the consumption of services. To maintain accuracy of operations, speed of update is critical. For data which is frequently updated, such as the location of busses or the present energy demand, redis is suitable. If you require more detailed data, such as history of infrastructure assets or environmental monitoring logs, then MongoDB has more to offer.

6.1 Comparison of Update Time



MongoDB (left) vs RedisDB (right)

Comparisons of update performance of MongoDB and Redis show that better performance was achieved in the experiment with Redis. MongoDB took 17.0093 milliseconds to update, while Redis took 12.46 milliseconds. This means that Redis is about 4.55 milliseconds quicker. This is because Redis is in-memory (RAM) only, which enables very fast read and write operations. By contrast, MongoDB is a document-based database, in which the data is generally stored on disk, and, thus, its update performance is somewhat less than that of relational databases because it incurs an extra layer of processing. This makes Redis more suitable if the application requires the need to fetch high speed updates in real-time, while MongoDB is more suitable if large structured data needs to be saved for a long time.

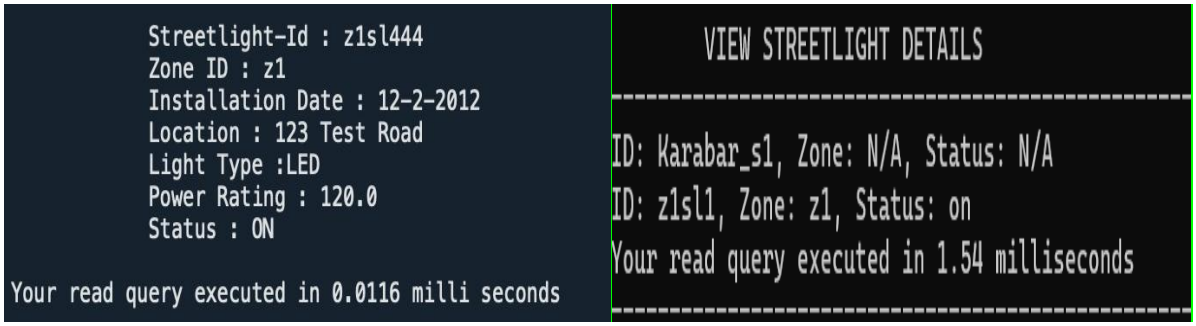
Resource usage includes CPU, memory, storage and network usage. This is crucial as sustainable systems should not expend computing resources in a wasteful manner. As Redis keeps data in memory, it is a very memory-intensive system and, in a way, it offers high throughput to the cost of memory. If you are storing large amounts of data over an extended period of time, then it's better to use MongoDB,

which stores data on disk and uses memory caching. So, Redis can be used to store data which can be lost easily and retrieved, while MongoDB can be used to store in memory and more complex queries.

Other evaluation concepts that are important include availability and scalability. Smart city systems need to be available as they can be used to assist with critical functions, like traffic management, energy monitoring and emergency notifications. For large workloads, MongoDB offers horizontal scaling and high availability options. In addition, Redis can be used to support high-performance workloads and fast response requirements with its high speed and fast response capabilities.

When evaluating, don't only consider speed of MongoDB and Redis. They are to be judged by their likeness with the system purpose. MongoDB is a good choice for structured sustainability data, such as sensor history, infrastructure assets and consumption data. Redis can be used for Live data, Caching, Alert Data and Temporary Values. They both have a design of the smart city platform of which MongoDB is used for long-term data storage, and Redis is used for real time performance.

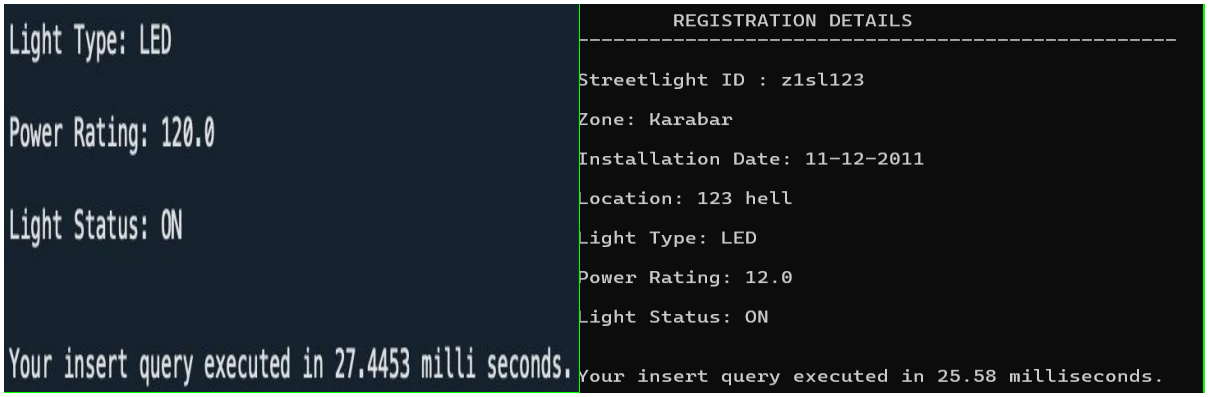
6.2 Comparison of Read



MongoDB (left) vs RedisDB (right)

In this experiment, the performance of the operations that have been performed on both the databases (MongoDB vs Redis), are shown to be much faster for MongoDB than for Redis. The time it took the Redis read operation is 1.54 milliseconds, while the MongoDB operation is very small at 0.0116 milliseconds. This means that in this particular test, the retrievals were much quicker from MongoDB. This could be because that document was quickly found in the database by using indexing or optimized query execution. While Redis is well known for its speed of in-memory data, this observation might have been influenced by the experimental setup, the data structure used, or the method used to implement it. For the read operation, MongoDB performed better in terms of read performance based on this test result, when compared to Redis.

6.3 Comparison of Insert Time



MongoDB (left) vs RedisDB (right)

This experiment benchmarks the speed of reading data from MongoDB and Redis, and the outcome is that the former is much faster in this case. As you can see, the read operation was very quick, 0.0116 milliseconds for MongoDB, while 1.54 milliseconds for Redis. In this specific test, it's clear that MongoDB is much faster at retrieving the data. This could be because MongoDB was able to efficiently access the requested document using an optimized query or indexing. Redis is often considered as an in-memory database, but in this case it may be the data structure, the experimental setup or the implementation technique that is the cause of this slow read speed. So, according to this test result, MongoDB was better in terms of read performance than Redis in the read operation.

Performance evaluation gives a general indication of the ability of a database to be used efficiently to support sustainable urban systems. Some metrics such as Insertion time, Retrieval time, Update time and resource usage directly affect the outcomes of system reliability, cost and sustainability.

(ALI et al., 2023)

In connection with our case study

For our Smart Streetlight Management System case study, a performance evaluation session will be held in order to compare the performance of MongoDB with Redis, on the following key performance indicators data insertion speed, retrieval time, update efficiency, resource usage, scalability, and availability. The system ingests the most recent smart city information, such as street light status and energy consumption, and their maintenance record, which is crucial to perform an evaluation with realistic workloads to assess the capacity of each database to sustain smart city infrastructure while keeping sustainability. They will be benchmarked across structured data to be stored flexibly over time in MongoDB and in-memory performance for real-time operations and alert handling in Redis. The evaluation will help to identify which NoSQL database technology is suitable for a smart streetlight management environment to enable SDG 7 (Affordable and Clean Energy) and SDG 11 (Sustainable Cities and Communities).

Data insertion time for MongoDB and Redis: In the Smart Streetlight Management System, the generated data from the street lights is real-time, and the time required for data insertion into a database is also a key requirement. The performance of Redis vs. Memcached will be compared with Redis being

expected to be faster for insertion time due to being stored in memory, thus enabling near instant write operations, making it ideal for live sensor data and instant updates. Structured or semi-structured data can also be inserted easily, especially when you need to insert a lot, although the data is stored in the main memory, it might be slightly slower than Redis. The evaluation of insertion times in both databases is used to verify the most appropriate database to efficiently run smart city applications in real time.

References

Carpentier, C. L., & Braun, H. (2020). Agenda 2030 for Sustainable Development: A powerful global framework. *Journal of the International Council for Small Business*, *1*(1), 14-23.

Costa, D. G., Bittencourt, J. C. N., Oliveira, F., Peixoto, J. P. J., & Jesus, T. C. (2024). Achieving sustainable smart cities through geospatial data-driven approaches. *Sustainability*, *16*(2), 640.

El Khatib, M. (2023). Performance Evaluation of PostgreSQL, MongoDB and Redis for Embedded Driver Monitoring Systems: Benchmarking PostgreSQL, MongoDB and Redis for Real-Time Driver Monitoring on a Raspberry Pi.

Hasan, M. A., Ahmed, F., Chowdhury, A. H., Leion, M. K., & Prince, M. R. F. (2022). *Analysis on the performance comparison between a relational database and a non-relational database for big data application* (Doctoral dissertation, Brac University).

Marat Nuriev, Rimma Zaripova, Yanova, O., Koshkina, I., & Andrey Chupaev. (2024). Enhancing MongoDB query performance through index optimization. *E3S Web of Conferences*, *531*, 03022–03022. https://doi.org/10.1051/e3sconf/202453103022

MongoDB. (n.d.-a). Data modeling in MongoDB. MongoDB Documentation. https://www.mongodb.com/docs/manual/data-modeling/

MongoDB. (n.d.-b). Schema design process. MongoDB Documentation. https://www.mongodb.com/docs/manual/data-modeling/schema-design-process/

MongoDB. (n.d.-c). Schema validation. MongoDB Documentation. https://www.mongodb.com/docs/manual/core/schema-validation/

MongoDB. (n.d.-d). Data modeling best practices. MongoDB Documentation. https://www.mongodb.com/docs/manual/data-modeling/best-practices/

Redis. (n.d.-a). Redis data types. Redis Documentation. https://redis.io/docs/latest/develop/data-types/

Redis. (n.d.-b). Redis sorted sets. Redis Documentation. https://redis.io/docs/latest/develop/data-types/sorted-sets/

The Global Goals. (n.d.). Goal 11: Sustainable cities and communities. https://globalgoals.org/goals/11-sustainable-cities-and-communities/

United Nations. (n.d.). Goal 7: Affordable and clean energy. United Nations Sustainable Development. https://www.un.org/sustainabledevelopment/energy/