Eco Bike Hubs Case Study

Bradley Watkins

January 2025

Abstract

This report provides an overview of the differences between SQL and NoSQL databases. Also discussed here is a case study on Eco Bike Hubs, including performance tuning suggestions and transaction implementation to solve issues raised by the company.

Contents

1 SQL vs NoSQL **2**

1.1 SQL 2

1.2 NoSQL 3

1.3 Conclusion 4

2 Performance Tuning **4**

2.1 Database Design 4

2.2 Secondary Index 1 5

2.3 Secondary Index 2 6

2.4 Secondary Index 3 6

3 Transactions **6**

3.1 Transaction 1 7

3.2 Transaction 2 7

4 Conclusion **7**

1 SQL vs NoSQL

Pronto Parcels have requested clarification on the difference between SQL and NoSQL databases. This section will discuss the main characteristics of each and a brief summary of their use cases.

1.1 SQL

Structured Query Language (SQL) is a declarative programming language used for managing and manipulating relational databases. Relational databases store structured data in a schema, which models entities and their relationships [1]. Entities are stored in tables consisting of rows and columns, with constraints placed on the data to ensure integrity [2]. Relational database tables are typically normalised to increase efficiency by breaking data into related tables, which can be joined later to view the complete data set [3].

SQL provides a mechanism for defining relational databases and their schema, with standardised operations for querying and manipulating the data within those databases. Relational databases built with SQL are highly reliable due to enforcing ACID properties (Atomicity, Consistency, Isolation, Durability). ACID properties ensure that data is timely and valid, has high integrity, and is able to be recovered in the event of a system failure [4].

Performance of SQL operations is typically high, allowing users to create complex queries against large volumes of data. However, database performance can degrade when complex queries like joins and aggregations are executed without proper optimisation [5]. One of the ways in which relational databases can be optimised is by creating secondary indexes on commonly queried columns [6]. Indexes serve as quick lookup mechanisms for the database query optimiser, reducing the number of pages searched to find the required data. This can greatly improve the performance of common lookup queries but may impact write performance, so indexes must be carefully selected [7].

Relational databases are the most commonly used storage system [8], but there are many challenges that must be considered when working with them. Increasing data volumes can lead to performance issues, and scaling relational databases can be difficult and costly when increasing hardware performance on the server [9]. These costs can be prohibitive for some applications, with limited horizontal scaling options due to the impact on ACID compliance [9, 4]. The complexity of relational databases can also lead to antipatterns arising in their design, which can make maintenance and change difficult [10]. Another common issue is SQL injection attacks, where attackers insert malicious code into SQL statements to manipulate the database. These attacks are considered one of the top ten vulnerabilities to online business, so awareness of ways to combat these attacks

is critical to success [11].

1.2 NoSQL

Not only SQL (NoSQL) databases are a classification of databases that deviate from the traditional relational database model, offering a variety of data storage methods, including document, key-value, column-family, and graph databases, each catering to different requirements [2]. NoSQL databases are designed to be highly flexible, often employing a schema-less or flexible schema design that allows for dynamic data storage without the need for a predefined schema [8]. This flexibility allows for easy horizontal scaling, with NoSQL databases able to distribute data across multiple servers to increase performance and availability [1]. By using a shared-nothing approach, NoSQL databases achieve high fault tolerance as data can be retrieved from any available server, even if some of the servers have failed [12]. Typically, NoSQL databases have lower hardware demands, enabling cost-effective horizontal scaling compared to relational databases, which usually require more complex and expensive scaling solutions [9].

While relational databases enforce ACID properties, NoSQL databases often adopt BASE properties (Basically Available, Soft State, Eventually Consistent). BASE properties provide less strict guarantees than ACID, instead focusing on availability and partition tolerance over consistency. Eventual consistency allows data to be temporarily inconsistent across the system, with the assurance that it will eventually converge to a consistent state. This means that some parts of the system may display stale data when queried, which may not be acceptable in scenarios requiring immediate consistency, such as real-time financial transactions or live sports updates [1]. A key concept related to NoSQL databases is the CAP theorem (Consistency, Availability, Partition Tolerance), which asserts that a distributed system can only guarantee two out of the three properties simultaneously [2]. The priorities of a given system must be carefully considered when making the choice to use a NoSQL model.

NoSQL databases are well-suited for systems that demand high performance and handle large volumes of data, often outperforming relational databases in such environments due to their ability to scale horizontally and efficiently handle unstructured data [9]. While NoSQL databases offer many advantages, they also come with drawbacks, such as weaker enforcement of data integrity and consistency compared to relational databases [8]. Therefore, NoSQL may not be suitable for use cases that require strict ACID compliance, like financial transactions [4].

1.3 Conclusion

SQL and NoSQL databases both have their strengths and weaknesses, making the choice between them dependent on the given use case.

SQL databases are ideal for systems that require complex queries, high data integrity, and transactional integrity thanks to their ACID properties [4]. As the most commonly used storage system [8], SQL databases are well-suited for use cases where relationships between entities are understood and consistent, such as financial services and resource management [2].

NoSQL databases offer greater flexibility and the ability to easily scale in distributed environments [1, 8]. They are well-suited for systems that expect high volumes of data throughput and where availability and fault tolerance are desirable [12]. Example systems may include social media platforms, real-time web applications, and big data analytics.

The choice between SQL and NoSQL is not trivial, but by carefully considering requirements such as scalability, data structure, and consistency, a valid choice can be made. Pronto Parcels should review the needs of their system and consider the trade-offs between SQL and NoSQL databases to make an informed decision.

2 Performance Tuning

Eco Bike Hubs have requested an investigation into their database query performance and potential improvements. This section presents three suggested secondary indexes on columns and the rationale behind their selection.

2.1 Database Design

Eco Bike Hubs use a relational database to manage their business data. There are two main data flows through the system:

- Management of customer bookings, including bicycle rentals, service, repair, and sales.
- Management of inventory trading.

Figure 1: Customer booking flow via UML

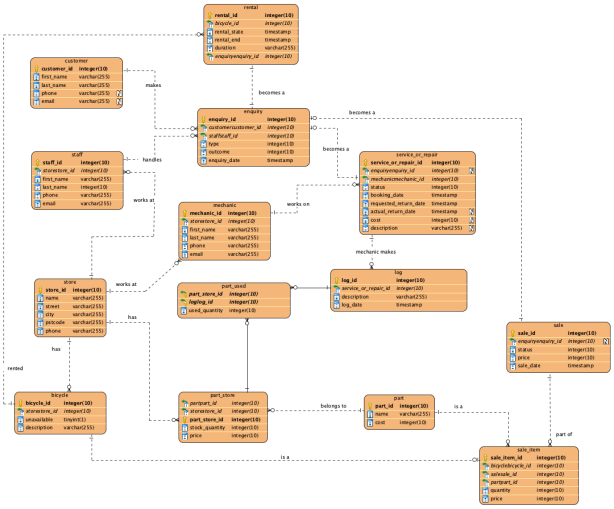

Figure 1 illustrates the database design to facilitate the customer booking flow. Initially, the customer makes an enquiry with staff at a store, which either leads to an abandoned enquiry or a booking that is subsequently handled. Booking types include rentals, service or repair, and sales, with each type requiring different information to be stored.

Figure 2: Inventory trading flow via UML

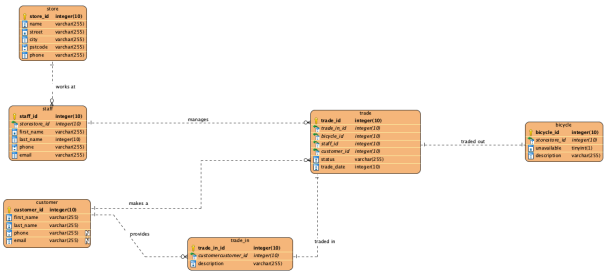

Figure 2 illustrates the database design to facilitate the customer trade flow. The customer trade will be handled by a member of staff who will either accept or reject the trade.

2.2 Secondary Index 1

```
CREATE INDEX bicycle_description ON bicycle(description);
```

The first suggestion for a secondary index is on the `description` column in the `bicycle` table. This column is frequently queried in both data flows, especially for customer trades, sales, and rentals. When a customer

and staff member work together to find a suitable bicycle, they likely have key features in mind to compare against this column. Creating an index on this column will enable faster lookups and improve the performance of these queries.

2.3 Secondary Index 2

```
CREATE INDEX part_name ON part(name);
```

The second suggestion for a secondary index is on the `name` column in the `part` table. This column is commonly queried for both customer bookings and internal staff use. When a customer seeks to purchase a custom-built bike or individual parts, they search for specific parts by name or parts that match a description. Similarly, internal staff, such as mechanics working on a service or repair, query this column to ensure necessary stock is available. Additionally, this column is frequently used in joins with other tables, such as the `part_store` table. Customers and staff query parts at specific stores, making this join necessary to retrieve the desired information. Creating a secondary index on this column will enhance query performance throughout the business.

2.4 Secondary Index 3

```
CREATE INDEX rental_period ON rental(rental_start, rental_end);
```

The third and final suggested secondary index is on the `rental_start` and `rental_end` columns of the `rental` table. These columns are crucial for managing bicycle availability for rental, as no two rentals for the same bicycle can overlap. These columns frequently appear in queries regarding bicycle availability for rental, sale, and trade. A sale or trade cannot occur before the final rental concludes and the bicycle is returned to the store. Indexing these columns will enhance both query and insert performance since they frequently appear in functions and triggers.

3 Transactions

As part of the database design implemented for Eco Bike Hubs, two example transactions have been created to solve problems raised by the company. This section will cover these transactions and the ways in which they combat the issues raised.

3.1 Transaction 1

It was noted that bicycle repair bookings were sometimes recorded without the specific return dates requested by customers. To combat this, a transaction covering the customer enquiry and booking process was implemented. This transaction ensures that the requested return date is present in the insert statement and that it is recorded in the system. By using a transaction, the system can guarantee that the requested return date is recorded, preventing the issue from occurring. In the case that there is an error, a roll back will occur, ensuring that the system remains as it was prior to the transaction and allowing the process to be retried. See Appendix Transaction 1 for the full transaction code.

3.2 Transaction 2

The second issue raised by Eco Bike Hubs was that customers were able to book bicycle hires that were not valid. The transaction implemented to combat this issue makes use of locks to ensure that race conditions cannot occur, preventing the issue. By locking the `rental` table, the system prevents simultaneous rental bookings occurring, ensuring that checks done against available dates are timely and correct. The lock on the `bicycle` table prevents the bicycle from being sold or traded prior to the current rental booking being completed. These locks ensure that the system remains consistent and that data integrity is maintained. See Appendix Transaction 2 for the full transaction code.

4 Conclusion

The first section of this report covered the differences between SQL and NoSQL databases, providing a comparison of their characteristics and use cases. Structured Query Language (SQL) is a specialised declarative programming language for defining and manipulating relational databases. Relational databases are highly reliable due to the enforcement of ACID properties, making them ideal for systems requiring data consistency and integrity. They can, however, be difficult to scale and maintain, with performance issues arising as data volumes increase.

Not Only SQL (NoSQL) databases come in a variety of forms, each offering different data storage methods. In general, NoSQL databases are more flexible and horizontally scalable than SQL databases, making them ideal for systems that handle large volumes of data and require high availability. However, NoSQL databases may not be suitable for systems requiring strict ACID compliance.

The second section of this report covered a case study on Eco Bike Hubs. The case study focused on the

design and implementation of a database and suggestions for secondary indexes that improve performance in critical areas. Additionally, two example transactions were provided to combat issues raised by Eco Bike Hubs.

References

[1] T. N. Khasawneh, M. H. AL-Sahlee, and A. A. Safia, "Sql, newsql, and nosql databases: A comparative survey," in *2020 11th International Conference on Information and Communication Systems (ICICS)*. IEEE, Apr. 2020, pp. 013–021.

[2] S. Ramzan, I. S. Bajwa, B. Ramzan, and W. Anwar, "Intelligent data engineering for migration to nosql based secure environments," *IEEE Access*, vol. 7, pp. 69 042–69 057, 2019.

[3] A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva, and U. Saxena, "Nosql databases: Critical analysis and comparison," in *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*. IEEE, Oct. 2017, pp. 293–299.

[4] J. Pokorny, "Nosql databases: a step to database scalability in web environment," *International Journal of Web Information Systems*, vol. 9, no. 1, pp. 69–82, Mar. 2013.

[5] M. Nuriev, R. Zaripova, A. Sinicin, A. Chupaev, and M. Shkinderov, "Enhancing database performance through sql optimization, parallel processing and gpu integration," *BIO Web of Conferences*, vol. 113, p. 04010, 2024.

[6] I. C. Saidu, M. Yusuf, F. C. Nemariyi, and A. C. George, "Indexing techniques and structured queries for relational databases management systems," *Journal of the Nigerian Society of Physical Sciences*, p. 2155, Sep. 2024.

[7] A. Arteta Albert, N. Gómez Blas, and L. F. d. Mingo López, "Intelligent indexing—boosting performance in database applications by recognizing index patterns," *Electronics*, vol. 9, no. 9, p. 1348, Aug. 2020.

[8] A. A. Imam, S. Basri, R. Ahmad, J. Watada, and M. T. González-Aparicio, "Automatic schema suggestion model for nosql document-stores databases," *Journal of Big Data*, vol. 5, no. 1, Dec. 2018.

[9] S. Chickerur, A. Goudar, and A. Kinnerkar, "Comparison of relational database with document-oriented database (mongodb) for big data applications," in *2015 8th International Conference on Advanced Software Engineering & Its Applications (ASEA)*. IEEE, Nov. 2015, pp. 41–47.

[10] D. A. d. Costa, N. Grattan, N. Stanger, and S. A. Licorish, "Studying the characteristics of sql-related development tasks: An empirical study," *Empirical Software Engineering*, vol. 28, no. 3, Apr. 2023.

[11] D. A. Kindy and A.-S. K. Pathan, "A detailed survey on various aspects of sql injection in web applications: Vulnerabilities, innovative attacks, and remedies," *International Journal of Communication Networks and Information Security (IJCNIS)*, vol. 5, no. 2, Apr. 2022.

[12] D. R. Zmaranda, C. I. Moisi, C. A. Győrödi, R. S. Győrödi, and L. Bandici, "An analysis of the performance and configuration features of mysql document store and elasticsearch as an alternative backend in a data replication solution," *Applied Sciences*, vol. 11, no. 24, p. 11590, Dec. 2021.

Appendix

Transaction 1

Note: This transaction contains explicit id values for the enquiry and service or repair records. These values should be generated dynamically in a production environment to avoid conflicts.

```
DO $$
DECLARE
    new_enquiry_id INT;
    new_service_or_repair_id INT;
    requested_return_date TIMESTAMP := NOW() + INTERVAL '1 week';
    requested_in_system TIMESTAMP;
BEGIN
    -- Insert a new repair enquiry
    INSERT INTO enquiry (enquiry_id, customer_id, staff_id, enquiry_date, type, outcome)
    VALUES (100, 1, 1, NOW(), 'repair', 'booked')
    RETURNING enquiry_id INTO new_enquiry_id;

    -- Ensure that requested return date exists prior to insert
    IF requested_return_date IS NULL THEN
        RAISE EXCEPTION 'Requested return date is not present';
```

```
    END IF;

    -- Create a new service_or_repair record for the enquiry
    INSERT INTO service_or_repair (service_or_repair_id, enquiry_id, mechanic_id, booking_date, requested
    VALUES (100, new_enquiry_id, 1, NOW(), requested_return_date, 'in progress', 'Replace brake pads and
    RETURNING service_or_repair_id INTO new_service_or_repair_id;

    -- Ensure requested return date has been recorded in the system
    SELECT service_or_repair.requested_return_date INTO requested_in_system
    FROM service_or_repair
    WHERE service_or_repair_id = new_service_or_repair_id;

    if requested_in_system IS NULL THEN
        RAISE EXCEPTION 'Requested return date not recorded in the system';
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        RAISE NOTICE 'Transaction rolled back due to error: %', SQLERRM;
    RAISE;
END;
$$;
```

Transaction 2

Note: This transaction contains explicit id values for the enquiry and rental records. These values should be generated dynamically in a production environment to avoid conflicts.

```
DO $$
DECLARE
    new_enquiry_id INT;
    new_rental_id INT;
    rental_start TIMESTAMP := '2025-01-16 10:00:00';
    rental_end TIMESTAMP := '2025-01-16 20:00:00';
```

```
BEGIN
    -- Lock the rental table to prevent race condition on rental times
    LOCK TABLE rental IN EXCLUSIVE MODE;
    -- Lock the bicycle table to prevent trades and sales while booking a rental
    LOCK TABLE bicycle IN EXCLUSIVE MODE;

    -- Insert a new rental enquiry
    INSERT INTO enquiry (enquiry_id, customer_id, staff_id, enquiry_date, type, outcome)
    VALUES (101, 1, 1, NOW(), 'rental', 'booked')
    RETURNING enquiry_id INTO new_enquiry_id;

    -- Insert a new rental record
    INSERT INTO rental (rental_id, enquiry_id, bicycle_id, duration, rental_start, rental_end)
    VALUES (101, new_enquiry_id, 5, 'full-day', rental_start, rental_end)
    RETURNING rental_id INTO new_rental_id;
EXCEPTION
    WHEN OTHERS THEN
        RAISE NOTICE 'Transaction rolled back due to error: %', SQLERRM;
    RAISE;
END;
$$;
```