EXECUTIVE SUMMARY FOR
HOTEL MANAGEMENT SYSTEM DATABASE DESIGN FOR
DREAMLAND MANOR

TSM/ES/v3.0

BY

TSM CONSULTING FIRM

ASSIGNMENT: Development Team Project: Executive Summary

TEAM 3
Tasweem Beelunkhan, Shaun Bell-Gibson, Abiodun Maborukoje

July 2023

TABLE OF CONTENT

1. EXECUTIVE SUMMARY

This executive summary analyses the proposed database design for the Hotel Management System (HMS) of Dreamland Manor. This summary aims to inform the organisation of the key findings and justification of the decisions made.

2. INTRODUCTION

A Hotel Management System (HMS) allows for various management operations including room allocation, payments and information relating to the guests' needs (Hsu & Hsu, 2011). The purpose of an HMS is to enhance operational efficiency and optimise the performance of the hospitality establishment by enabling a centralised platform to streamline processes (Kasavana & Cahill, 2014). TSM Consulting's role is to evaluate the needs of Dreamland Manor and create an HMS that is tailored to best suit these requirements.

3. SUMMARY OF METHOD

The design and development of the Hotel Management System (HMS) database have been successfully completed, following a systematic approach that encompassed various stages, including database planning, system definition, requirements gathering, evaluation of the data model, database modelling, data integration, and testing. During the planning stage, the project's objectives were defined, focusing on designing a database system to efficiently and effectively manage hotel operations. System definition was documented by identifying major system actors and business service areas, such as managers, staff, guests, and services like room reservation and booking.

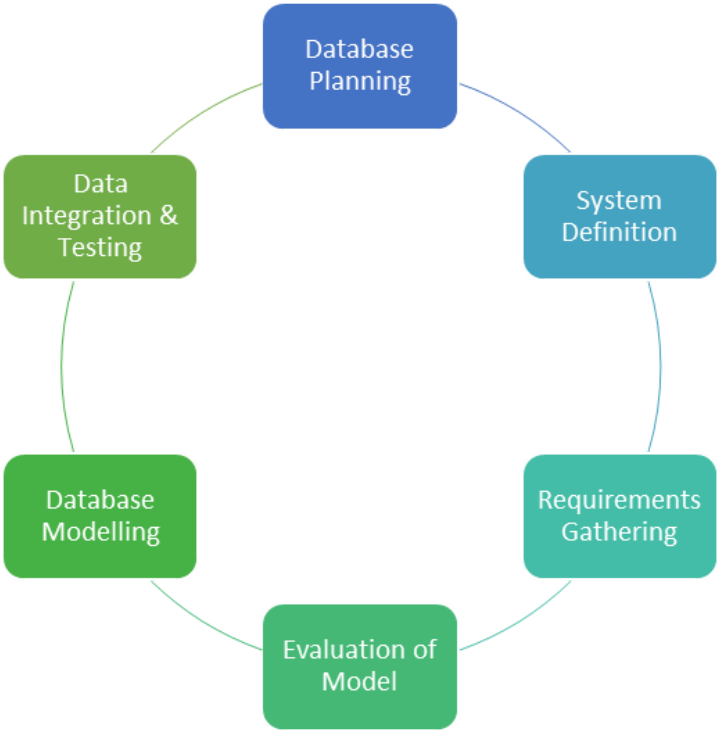


Figure 3-1 Design Approach

To gather the database requirements, fact-finding techniques such as interviews, observation, and research were explored (Connolly & Begg 2015). Desk research was primarily utilized by reviewing related work, including the analysis of functional hotel management systems by Yang (2013). The design report submitted earlier contains documentation of the functional and non-functional requirements. Based on these requirements and research findings, an object-based data model approach was adopted for database modelling (Connolly & Begg 2015). The identified information during the requirement phase was organized into logical entities, representing distinct concepts such as Hotel, Rooms, Amenities, Users, Gallery, Reservations, Billing, and Identities. Specific attributes were determined for each entity, defining appropriate data types and sizes. Primary keys were established to ensure record uniqueness and relationships between tables were established using foreign keys.

To enhance understanding and facilitate clear communication with stakeholders, the logical database design was documented by mapping entities, attributes, and relationships into a logical data model. The chosen data model for the HMS is the Entity-Relationship Model (ER

Model), known for its effectiveness in database design (Song & Froehlich, 1995). The ER Model aligns well with the database requirements and offers several strengths. Throughout the project, The team reviewed and refine the database design, ensuring accuracy, completeness, and alignment with the requirements. Factors like data redundancy, consistency, and data integrity were carefully considered. Normalization rules, including First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF), were applied to organize the database effectively, minimizing data redundancy.

In the post-design phase, the physical database was developed using MySQL and Python. The Mysql.connector Python package was used to create and design the database, which was hosted locally using MySQL Community Server and Workbench. Data Definition Language (DDL) statements were utilized to create tables, define columns, and establish relationships. Representative test data sourced from Kaggle (Mostipak, J. et al. 2019) and the Faker Python package were used to populate the database for realistic testing. Rigorous testing was conducted using Data Manipulation Language (DML) statements to perform operations like data insertion, update, and retrieval. Scenarios such as room reservations, guest information retrieval, billing calculations, and data integrity checks were extensively tested to ensure the functionality, accuracy, and reliability of the database. Identified issues and errors were promptly addressed to ensure the smooth functioning of the database.

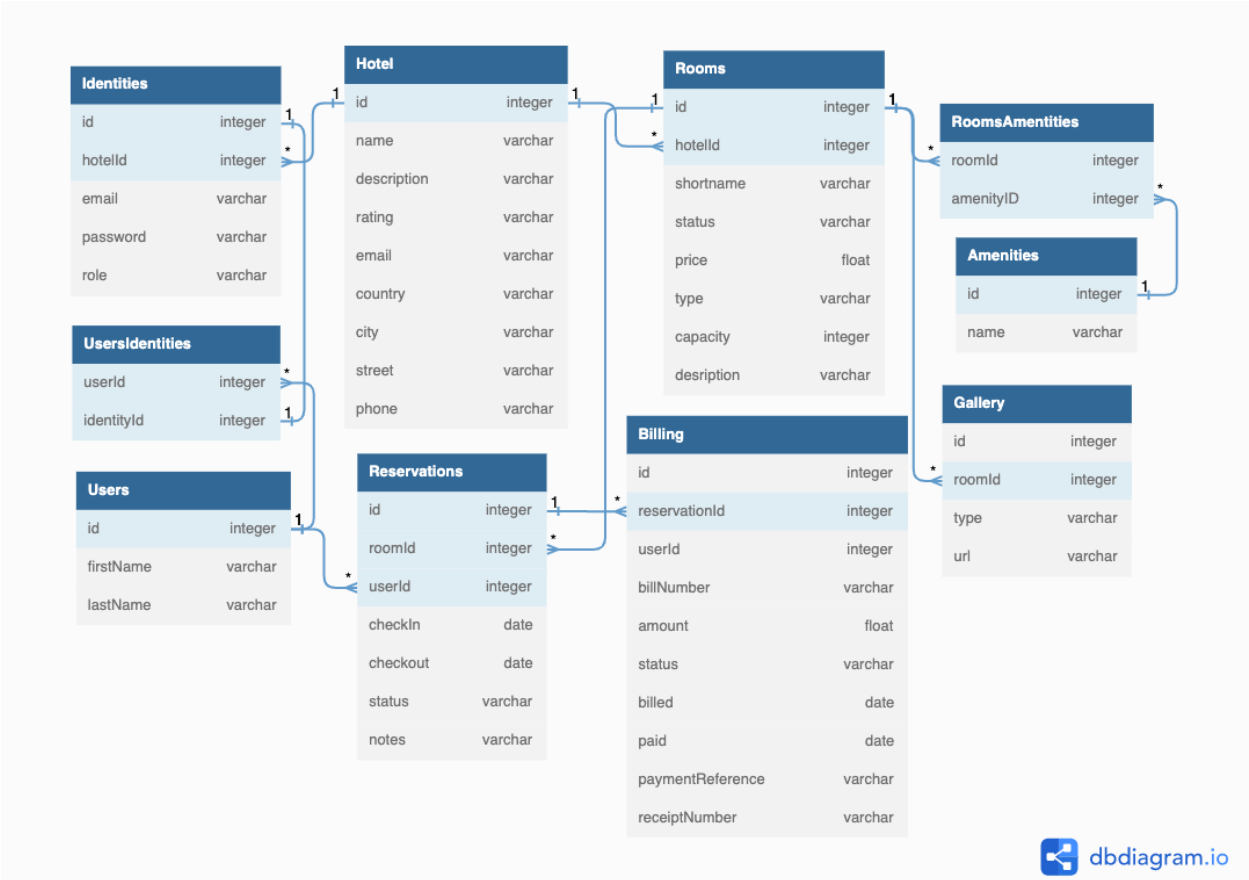


Figure 3.2 The ER Diagram representing the logical data model of the proposed Dreamland Manor HMS Database

4. SUMMARY FINDINGS

Our project's objective was to create a logical database that would give the business a more effective, scalable, and reliable data management solution. We set out to change the redundant and inconsistent existing data architecture into a system that streamlines information storage and retrieval while maintaining data integrity.

1. The relational database model provides a standardised method for data storage, making it easier to access and manipulate information. The relational model's structured style, which relies on tables with rows and columns, gave the data a high level of organisation and efficient relationships thus allowing an increased overall operational efficiency (Elmasri & Navathe, 2011).

2. Also, since the database design demonstrates a well-defined structure, it follows best practices for normalisation. The tables are appropriately organised, with minimal data redundancy and efficient relationships. Without an appropriate HMS system, the same data would be repeated in multiple places, leading to higher storage costs and a greater potential for errors during data input. For example, instead of repeating guest information (like name, contact, etc.) in every booking record, it's stored in a separate 'Guest' table, and only the unique 'Guest ID' is referred to in the 'Booking' table. Thus, this leads to a reduction in the volume of duplicate data significantly, leading to cost savings and improved data integrity.

3. Another key finding is that the developed database system demonstrated a notable improvement in data consistency. This is due to the relational database systems' intrinsic ACID features (Atomicity, Consistency, Isolation, Durability), which guarantee that the data will stay consistent even in the case of system failure or concurrent data operations (Gray & Reuter, 1993). "Atomicity" makes sure that all booking-related tasks are finished concurrently. After check-out, "consistency" upholds the accuracy of data, such as room availability status. Conflicts like double booking from concurrent reservations are avoided through "isolation". Last but not least, "Durability" ensures that completed transactions, such as confirmed bookings, endure even after system malfunctions, preventing data loss.

4. The relational model enhances data integrity using constraints that enforce business rules and accuracy. Notably, primary, and foreign keys restrict data entry into tables, minimising errors. Primary keys provide unique identifiers for each record, while foreign keys establish relationships between tables, ensuring consistent and reliable data. Thus, these constraints uphold data integrity, bolstering the overall reliability of the database.

For instance, each hotel room has been assigned a unique identifier or "primary key." This guarantees that no two rooms share the same identifier, thus eliminating the risk of double booking.

On the other hand, "foreign keys" could be used to link related information across different tables. For example, a booking record might have a reservation ID which is the guest as a foreign key, linking it to the guest information in a different table. This ensures that every booking record is associated with a valid guest, preventing errors like creating a booking for a non-existent guest.

5. The penetration test conducted on the MySQL database revealed the presence of SQL injection vulnerabilities (see Annex D). Recommendations were made to mitigate these risks while implementing the database user interface.

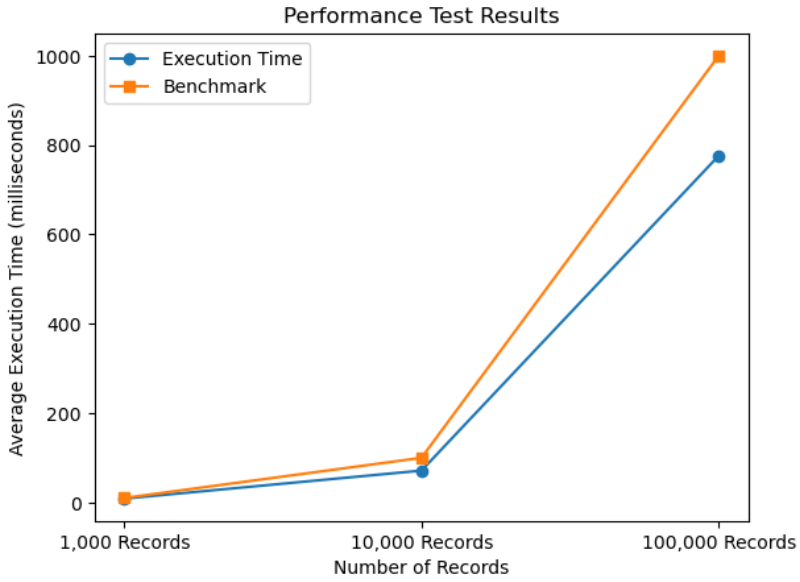


Figure 4.1 Result of the performance test carried out on the physical database, the test makes use of the Response Time Benchmark. The result shows that the database performs efficiently below the benchmark target set while fetching data from the database.

5. EVALUATION OF DATABASE PROPOSAL

Comparisons between SQL and NoSQL databases have shown that NoSQL databases can outperform SQL databases in a variety of parameters including being faster and more flexible (Nayak et al., 2013). However, not one NoSQL database consistently outperformed a SQL database in multiple parameters (Li & Manoharan, 2013); each NoSQL database has its limitations, may not be ACID (Atomicity, Consistency, Isolation, and Durability) compliant and can be difficult to maintain (Nayak et al., 2013). The limitations of SQL databases are mainly centred around being less flexible than the NoSQL alternatives making it difficult to adapt the existing schema when in place and forcing the data to adhere to a fixed tabular structure (MongoDB, nd). These limitations of the SQL databases have a greater effect on semi/unstructured data and therefore will have a limited effect on the structured data that will be used within this database (see Annex A).

Based on the requirements of Dreamland Manor, a relational database management system (RDBMS) such as MySQL would be a good choice. MySQL is a popular open-source RDBMS that provides various benefits that align with the requirements as stated in Suehring (2002):

- Data Integrity and Consistency
- Scalable
- High-Performance
- Robust Security
- Broad Compatibility and Large Ecosystem
- Cost-effectiveness (Open Source)
- Excellent support is available

In comparison to other RDBMS products MySQL can outperform by reaching higher speeds (Suehring, 2002). MySQL provides several security features to protect the personal data of customers including privilege commands and firewalls (Hamidi et al., 2022). Scalability is the main challenge when using MySQL, however, due to its simple syntax and mild complexity MySQL is easy to use and manage (Altexsoft, 2019). There are other RDBMSs available that offer more functionality. However, overall MySQL provides the greatest array of benefits (see Annex B) while balancing cost-effectiveness with functionality (Denton & Peace, 2003).
To protect the sensitive personal data of the customers the customer's names and bank details are secured in a separate table with access control, and each customer is assigned a unique pseudonym id number. Using a pseudonym maintains the ability to relate different records that relate to an individual to be linked even without storing the direct identifiers within the data which is highly beneficial for a relational database (Data Protection Commission, 2019).

To assess the performance and efficiency of the MySQL database, we considered various benchmarks, including Response Time, Throughput, and Load Test benchmarks (Nichter, 2021). Due to time and computing power limitations, the Response Time benchmark was selected, which measures the execution time of a specific operation or query. In this case, fetching (100,000) records from the "Reservations" table served as the benchmark scenario. Measuring the average execution time of approximately 0.78 seconds, obtained through 10 query runs using the "timeit" Python library (see Appendix Annex E). The table below measures the response time result against the study done by Power, R. (2009).

Table 5.1 Result of the performance test carried out on the physical database, the test makes use of the Response Time Benchmark.

Records	**MySQL Benchmark**	**HMS DB Result**
1,000	x < 10 milliseconds	8.57 milliseconds
10,000	x < 100 milliseconds	71.37 milliseconds
100,000	x < 1000 milliseconds	776.21 milliseconds
Test Machine Specification: Lenovo ThinkPad, Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz, 16GB RAM. Windows 11 Pro, MYSQL Community Edition.		

6. CONCLUSIONS & RECOMMENDATIONS

In conclusion, TSM Consulting successfully develop the Hotel Management System (HMS) database for Dreamland Manor. The project followed a systematic approach, delivering a well-structured relational database that ensures efficient data storage, improved data consistency, and data integrity. Adequate testing validated the functionality, accuracy, and reliability of the database. The evaluation of the database proposal identified MySQL as a suitable choice, offering benefits such as data integrity, scalability, high performance, robust security, broad compatibility, and cost-effectiveness. The designed HMS database provides Dreamland Manor with a reliable and efficient solution for managing its hotel operations while ensuring data integrity and security.

The following are recommendations for further improvement:

1. GDPR Recommendations:
   To maintain the highest level of security the GDPR (Burton - Article 32, 2020) recommends two policies: Regular software updates and security assessments. Regular software updates keep the database up to date with the latest patches and updates to address any known vulnerabilities. Regular security assessments including penetration testing and vulnerability scanning to proactively identify any security weakness in the database (Burton, 2020). If a breach of personal data occurs it must be reported to the supervisory authority in accordance with Article 55 (see annex B) within 72 hours (Burton - Article 33, 2020).

2. Backup Recommendation:
   Back up data with one of the best tools to secure the data in case of loss (Hamidi et al., 2022). The grandfather-father-son (GFS) backup method is a widely used rotation-based schedule that involves creating daily(son), weekly(father) and monthly(grandfather) backups. Advantages of the GFS system include the ability to be restored from the past week, any week in the past month or from any month since the system began (Oteng-Boateng, 2011). This range of backup options provides a greater range of options than other backup options but can require a significant number of tapes.

3. Security Recommendation:
   Implement preventive measures to address SQL injection vulnerabilities. This includes using parameterized queries (prepared statements) (Halfond and Orso 2005), validating and sanitizing input, adhering to the least privilege principle, following secure coding practices, regularly patching and updating the database management system, and conducting security audits and testing.

7. REFERENCES

Altexsoft (2019) *Comparing database management systems: Mysql, PostgreSQL, MSSQL server, mongodb, Elasticsearch, and others*, *AltexSoft*. Available at: https://www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/ (Accessed: 13 July 2023).

Burton, C. (2020) 'Article 32 security of processing', *The EU General Data Protection Regulation (GDPR)* [Preprint]. doi:10.1093/oso/9780198826491.003.0068.

Burton, C. (2020) 'Article 33 notification of a personal data breach to the Supervisory Authority', *The EU General Data Protection Regulation (GDPR)* [Preprint]. doi:10.1093/oso/9780198826491.003.0069.

Data Protection Commission (2019) *Guidance note - data protection commissioner*. Available at: https://www.dataprotection.ie/sites/default/files/uploads/2022-04/Anonymisation%20and%20Pseudonymisation%20-%20latest%20April%202022.pdf (Accessed: 06 July 2023).

Denton, J.W. & Peace, A.G. (2003) "Selection and Use of MySQL in a Database Management Course", Journal of Information Systems Education (JISE), 14(4), pp. 401–408.

EDPB (2023) *Guidelines 9/2022 on Personal data breach notification under GDPR*, *Guidelines 9/2022 on personal data breach notification under GDPR*. Available at: https://edpb.europa.eu/system/files/2023-04/edpb_guidelines_202209_personal_data_breach_notification_v2.0_en.pdf (Accessed: 12 July 2023).

Hamidi, A., Hamraz, A.R. and Rahmani, K. (2022) *Database Security Mechanisms in MySQL* [Preprint], (4). Available at: https://arj.af/index.php/arj/article/view/45 (Accessed: 2023).

Li, Y. and Manoharan, S. (2013) 'A performance comparison of SQL and NoSQL databases', *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)* [Preprint]. doi:10.1109/pacrim.2013.6625441.

MongoDB (nd) *Data Modeling Introduction*, *Data Modeling Introduction - MongoDB Manual*. Available at: https://www.mongodb.com/docs/manual/core/data-modeling-introduction/ (Accessed: 03 July 2023).

Nayak, A., Poriya, A. and Poojary, D. (2013) 'Type of NOSQL Databases and its Comparison with Relational Databases', *International Journal of Applied Information Systems (IJAIS)*, 5(4), pp.

16–19. Available at: https://research.ijais.org/volume5/number4/ijais12-450888.pdf (Accessed: 30 June 2023).

Oteng-Boateng, M.L. (2011) ‘ DATA BACKUP SECURITY: BEST PRACTICES FOR K-12 INTERNATIONAL SCHOOLS IN SOUTH KOREA ’, *LEWIS UNIVERSITY* [Preprint]. Available at: http://cs.lewisu.edu/mathcs/msis/projects/msis595_MonicaOtengBoateng.pdf.

Suehring, S. (2002) *MySQL Bible*. New York: Wiley. Available at: http://box.cs.istu.ru/public/docs/other/_New/Books/Data/DB/MySQL/MySQL%20Bible.pdf (Accessed: 29 June 2023).

Connolly, T. M. & Begg, C. E. (2015) Database Systems: A Practical Approach to Design, Implementation and Management. 6th ed. Essex: Pearson.

Yang, J. (2013) “Research and design of hotel management system model,” in Proceedings of the 2013 the International Conference on Education Technology and Information Systems. Paris, France: Atlantis Press.

Song, I. and Froehlich, K. (1995) "Entity-relationship modeling," in IEEE Potentials, vol. 13, no. 5, pp. 29-34, Dec. 1994-Jan. 1995, doi: 10.1109/45.464652.

Mostipak, J. et al. (2019) “Hotel booking demand,” Hotel Booking Demand Datasets. Available at: https://www.kaggle.com/datasets/jessemostipak/hotel-booking-demand (Accessed: May 7, 2023).

Power, R. (2009). The Oracle Set Query Benchmark. Available at: https://publications.csiro.au/rpr/download?pid=csiro:EP092149&dsid=DS1 (Accessed: July 13, 2023).

Nichter, D. (2021) Efficient MySQL performance: Best practices and techniques. Sebastopol, CA: O’Reilly Media.

Halfond, W. and Orso, A. (2005). AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks. 20th IEEE/ACM International Conference on Automated Software Engineering, ASE 2005. 174-183. 10.1145/1101908.1101935.

Elmasri, R. and Navathe, S.B. (2011) Database systems: Models, languages, design, and Application Programming. Boston, MA: Pearson.

Medjahed, B., Ouzzani, M. and Elmagarmid, A.K. (2009) ‘Generalization of acid properties’, Encyclopedia of Database Systems, pp. 1221–1222. doi:10.1007/978-0-387-39940-9_736.

Gray, J. and Reuter, A. (1993). Transaction Processing: Concepts and Techniques. Morgan Kaufmann.

Noor, Md. et al. (2011) 'An electronic intelligent hotel management system for international marketplace', International Journal of Advanced Computer Science and Applications, 2(3). doi:10.14569/ijacsa.2011.020316.

Putra, I.G., Kanca, I.N. and Wijaya, I.N. (2019) 'Online application of Hotel Management (Case Study the wing Ed Hotel of the bali state Politechnic)', Proceedings of the International Conference On Applied Science and Technology 2019 - Social Sciences Track (iCASTSS 2019) [Preprint]. doi:10.2991/icastss-19.2019.17.

8. APPENDIX

Annex A: Comparison of SQL vs NoSQL databases (Altexsoft, 2019)

RELATIONAL VS NON-RELATIONAL DATABASES

Feature	Relational databases (SQL)	Non-relational databases (NoSQL)

Data structure	• Organize data into tables with rows and columns
	• Strict schema
	• Data resides in records and attributes
		• Use different data models like
		  ○ document-oriented,
		  ○ key-value,
		  ○ graph,
		  ○ wide-column
		• Store unstructured data
		• No fixed schema

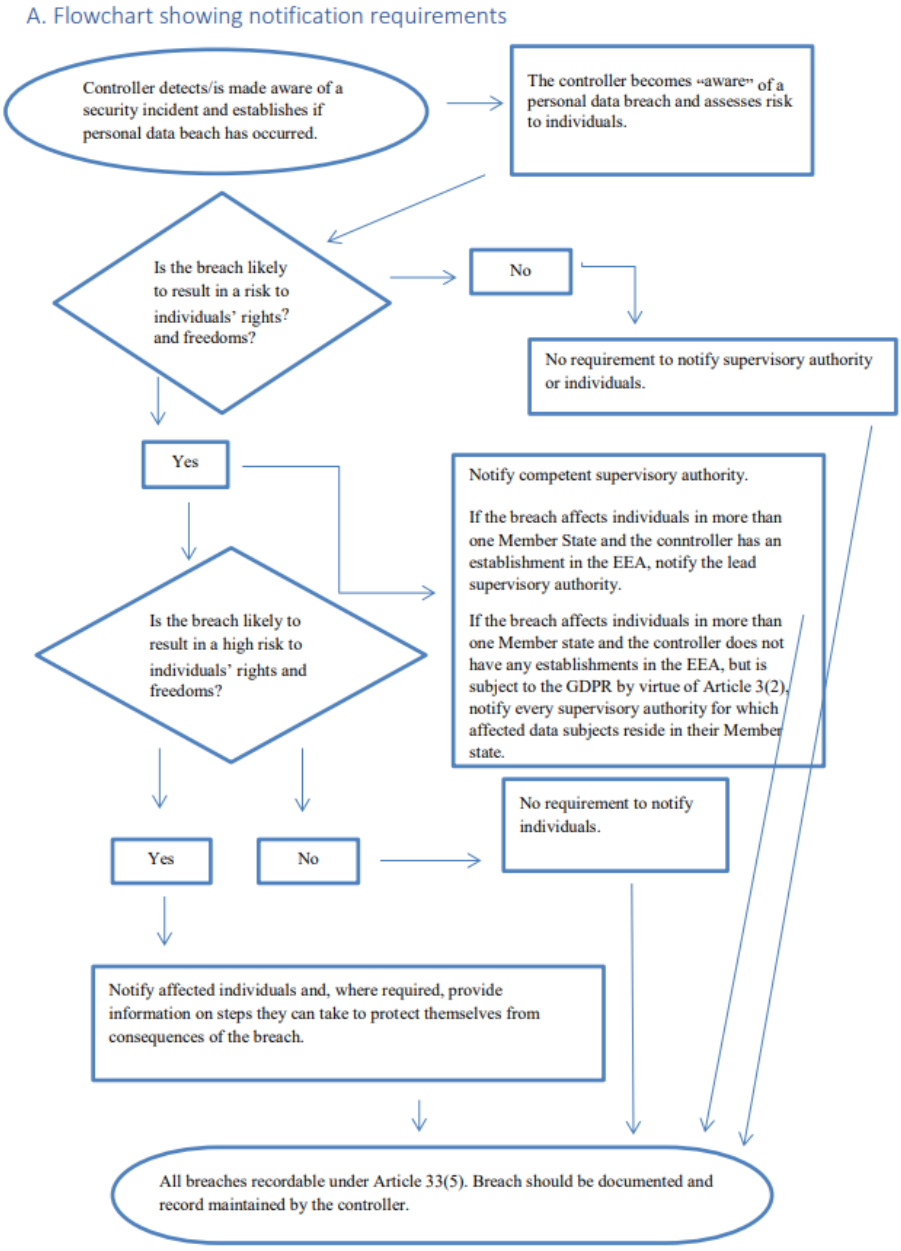
Language	Structured Query Language (SQL)	Various query languages depending on the data model
Scalability	• Scale vertically (more computer power to a single server) • Horizontal scaling is challenging and requires additional effort	• Scale horizontally (add more servers) • Share data between servers, decreasing the request-per-second rate in each server
Performance	• Perform well with intensive read/write operations on small to medium datasets • Can suffer when data and user requests grow	• High performance with distributed design • Provide simultaneous access to a large number of users • Store unlimited data sets in various formats
Security	• The integrated structure provides better security • ACID compliance is preferred for applications where database integrity is critical	• Generally weaker security • ACID guarantees are often limited to a single database partition • Some DBMSs offer advanced security features for compliance.
Use cases	Complex software solutions, eCommerce, financial applications	Storing and scaling unstructured data, MVPs for startups, sprint-based Agile development

Annex B: Comparison of DBMS (Altexsoft, 2019)

DATABASE MANAGEMENT SYSTEMS COMPARISON

	Database Type	Licensing	Scalability	Data Types Supported	Learning Curve
MySQL	SQL	GNU Generally Public License	Vertical, complex	Structured, semi-structured	Mild
MariaDB	SQL	GNU Generally Public License	Vertical	Structured, semi-structured	Mild
Oracle	Multi-model, SQL	Proprietary	Both (Vertical & Horizontal)	Structured, semi-structured, unstructured	Hard
PostgreSQL	Object-relational, SQL	Open-source	Vertical	Structured, semi-structured, unstructured	Hard
MSSQL	T-SQL	Proprietary	Vertical, complex	Structured, semi-structured, unstructured	Hard
SQLite	SQL	Public domain	Vertical	Structured, semi-structured, unstructured	Mild
MongoDB	NoSQL, document-oriented	SSPL	Horizontal	Structured, semi-structured, unstructured	Mild
Redis	NoSQL, key-value	Open-source, BSD 3-clause	Horizontal	Structured, semi-structured, unstructured	Mild
Cassandra	NoSQL, wide-column	Open-source	Horizontal	Structured, semi-structured, unstructured	Hard
Elasticsearch	NoSQL, document-oriented	Open-source	Horizontal	Structured, semi-structured, unstructured	Hard
Firebase	NoSQL, real-time database	Open-source	Horizontal	Structured, semi-structured, unstructured	Mild
Amazon DynamoDB	NoSQL, key-value	Proprietary	Horizontal	Structured, semi-structured, unstructured	Mild

Annex C: Notification Process Flowchart (EDPB, 2023)



Annex D: Penetration Test (SQL Injection)

In [95]:
```python
#Demostrating Vulnerability (directly embedding user input in SQL queries)
# Define the vulnerable parameter and payload
vulnerable_param = "shortname"
payload = "' OR '1'='1' -- "
# Construct and execute the SQL query with the payload
query = f"SELECT * FROM Rooms WHERE {vulnerable_param} = '{payload}'"
cursor.execute(query)
# Retrieve the results
results = cursor.fetchall()
# Check if the query returned any results
if results:
    print("SQL Injection vulnerability detected!")
    # Report the vulnerability and provide further details
else:
    print("No SQL Injection vulnerability detected.")
```

```
SQL Injection vulnerability detected!
```

In [100]:
```python
#Providing Solution to Prevent such Vulnerability (Parameterized Queries (Prepared Statements))
# Define the query template with placeholders
query = "SELECT * FROM Rooms WHERE shortname = %s"
# Execute the query with the user input as a parameter
roomname = "A"
#roomname = "' OR '1'='1' -- " #try running this will result in no records returned
cursor.execute(query, (roomname,))
# Retrieve the results
results = cursor.fetchall()
print("Room Records:")
for record in results:
    print(record)
```

```
Room Records:
(2, 1, 'A', 'Occupied', 177.0, 'Suite', 2, 'Individual through next stay raise author run.')
```

Annex E: Performance Evaluation (Response Time Benchmark)

In [94]:
```
#Performance Testing to fetch all records
# Define the test scenario: Example query execution
def perform_query():
    cursor.execute("SELECT * FROM Reservations LIMIT 100000")
# Measure execution time
execution_time = timeit.timeit(perform_query, number=10)  # Run the query 10 times
# Print the average execution time
print(f"Average execution time: {execution_time/10} seconds")
```

```
Average execution time: 0.7762015399996016 seconds
```

In [81]:
```
 #Performance Testing INSERTing 1000 record
# Define the test scenario: Example query execution
def perform_query():
    for _ in range(1000):
        # Insert statement for Reservations table
        reservation_room_id = 1
        reservation_user_id = 1
        reservation_checkin = row['arrival_date'] #fake.date_between(start_date='-1y', end_date='+1y')
        reservation_checkout = fake.date_between_dates(date_start=reservation_checkin, date_end=reservation_checkin+datetime.time
        reservation_status = row['reservation_status'] #fake.random_element(elements=('Confirmed', 'Pending', 'Cancelled'))
        reservation_notes = fake.sentence()
        cursor.execute(f"INSERT INTO Reservations (roomId, userId, checkin, checkout, status, notes) VALUES ('{reservation_room_i
# Measure execution time
execution_time = timeit.timeit(perform_query, number=10)  # Run the query 10 times
# Print the average execution time
print(f"Average execution time: {execution_time/10} seconds")
```

```
Average execution time: 0.8758083400000032 seconds
```

Annex F: Data Definition Language (DDL) Script

In [8]:
```
##Create tables
mycursor.execute("CREATE TABLE Hotel (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), description varchar(255), rating varc
mycursor.execute("CREATE TABLE Rooms (id INT AUTO_INCREMENT PRIMARY KEY, hotelId integer, shortname varchar(255), status varchar(
mycursor.execute("CREATE TABLE Amenities (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255))")
mycursor.execute("CREATE TABLE Users (id INT AUTO_INCREMENT PRIMARY KEY, firstName varchar(255), lastName varchar(255))")
mycursor.execute("CREATE TABLE Gallery (id INT AUTO_INCREMENT PRIMARY KEY, roomId integer, type varchar(255), url varchar(255))")
mycursor.execute("CREATE TABLE RoomsAmentities (roomid INT AUTO_INCREMENT PRIMARY KEY, amenityID integer)")
mycursor.execute("CREATE TABLE Reservations (id INT AUTO_INCREMENT PRIMARY KEY, roomId integer, userId integer, checkin date, che
mycursor.execute("CREATE TABLE UsersIdentities (userId INT AUTO_INCREMENT PRIMARY KEY, identityId integer, FOREIGN KEY (userId) F
mycursor.execute("CREATE TABLE Billing (id INT AUTO_INCREMENT PRIMARY KEY, reservationId integer, userId integer, billNumber varc
mycursor.execute("CREATE TABLE Identities (id INT AUTO_INCREMENT PRIMARY KEY, hotelId integer, email varchar(255), password varch

#Check tables were created
mycursor.execute("SHOW TABLES")
for x in mycursor:
  print(x)
```

In [9]:
```
mycursor.execute("ALTER TABLE Rooms ADD FOREIGN KEY (hotelId) REFERENCES Hotel (id)")
mycursor.execute("ALTER TABLE Identities ADD FOREIGN KEY (hotelId) REFERENCES Hotel (id)")
mycursor.execute("ALTER TABLE UsersIdentities ADD FOREIGN KEY (userId) REFERENCES Users (id)")
mycursor.execute("ALTER TABLE UsersIdentities ADD FOREIGN KEY (identityId) REFERENCES Identities (id)")
mycursor.execute("ALTER TABLE RoomsAmentities ADD FOREIGN KEY (roomId) REFERENCES Rooms (id)")
mycursor.execute("ALTER TABLE RoomsAmentities ADD FOREIGN KEY (amenityID) REFERENCES Amenities (id)")
mycursor.execute("ALTER TABLE Reservations ADD FOREIGN KEY (roomId) REFERENCES Rooms (id)")
mycursor.execute("ALTER TABLE Reservations ADD FOREIGN KEY (userId) REFERENCES Users (id)")
mycursor.execute("ALTER TABLE Billing ADD FOREIGN KEY (reservationId) REFERENCES Reservations (id)")
mycursor.execute("ALTER TABLE Gallery ADD FOREIGN KEY (roomId) REFERENCES Rooms (id)")
```