Technology Investigation Summary: Databases

# Potential Data to store:

* User accounts:
  + Login Credentials (Username, Passwords).
  + User-type (Host or Guest).
* Saved Host playlists:
  + Playlist title
  + List of songs, likely represented by IDs related to the music API.
* Jukebox Data:
  + Guests and Host connected per session
  + Chat logs per session
  + Songs in current session’s playlist
  + Songs in current session’s queue
  + Connection code and QR code
  + Jukebox location data:
    - Geographical location of jukeboxes for guests to find/locate (likely latitude and longitude or similar.)
    - Max connection range (in metres)
  + Token count per guest. Update based on token acquisition/spending.

# Software Database Requirements:

The Virtual Jukebox web-application will *likely require* the following database features   
(Derived from the Project Brief):

[Please note that these requirements were identified before the definition of a Software Requirements Specification]

* Confidentiality in the storage of user account details (Encryption, Authentication)
* Consistency and integrity in database operations and storage of user data (ACID transaction guarantees)
* Web-server compatibility.
* Ease of scalability, in the case that the application needs to upscale to store more types of data, and support more users.
* Efficiency for processing large volumes of data in short timeframes.

The application *may* benefit from these database features:

* Multi-modal data type support such as geospatial data, for storing jukebox location data.
* NoSQL functionality for addition of data that does not fit into a standard tabular RDBMS format (assists in future scalability).

Redundant or irrelevant database features for this use-case include:

* Data warehousing, as data analytics is not relevant for any Virtual-Jukebox function.
* Feature heavy systems (e.g. Block chain support, AI functionality), due to the lack of niche or complex functionality of the program.

# Database Comparisons:

Points derived and summarised from [1]-[5]:

## Oracle

Highly scalable, feature rich, proprietary RDBMS. One of the most popular RDBMS in use worldwide.

### Pros:

* Very scalable with high performance at large scales.
* Robust infrastructure
* Multimodal data-type support (e.g. JSON/XML, Spatial data)
* Block chain table support (tamper-proof, insert-only)
* Highly consistent with ACID transaction guarantees.

### Cons:

* Expensive commercial licensing, especially for smaller organisations
* Requires significant hardware resources to implement, and to maintain performance.

### Ideal Use-case:

* When data warehousing is desired for business analytics
* When working with very large databases is required, especially with multimodal data-types
* When ACID transaction guarantees are required.

## MySQL

Very popular RDBMS for web-applications. Focuses on speed and reliability over innovative features.

### Pros:

* Available free, but licenses with additional features are paid.
* Does not have intense hardware/resource requirements, but still scalable.
* Can be used alongside other database systems with ease
* Multiple user-interfaces available.

### Cons:

* Some features are not pre-defined such as incremental backups, and require manual implementation.
* Requires the use of a storage engine (e.g. InnoDB) for ACID compliance
* No built-in XML support
* Difficult to upscale (more steps required)

### Ideal Use-case:

* When Data reliability and speed are key.
* For small to medium web-applications

## MS SQL Server

Proprietary SQL RDBMS that works on both cloud and local servers, built for use on Windows systems, but is available for use on other operating systems.

### Pros:

* Dynamic data masking allows specification of how much sensitive data may be revealed
* Server-side scripting support
* Multi-modal data structures support
* Scalability features such as tooling to move on premise servers to the cloud.

### Cons:

* Expensive commercial licensing
* Resource intensive server technology

### Ideal Use-case:

* Businesses using full/majority Microsoft environments.
* To shift Microsoft SQL databases to the cloud
* When ACID transaction guarantees are required.

## PostgreSQL

Frequently used, open-source RDBMS for web-applications. Allows for the management of structured, and unstructured data using JSON at a large scale.

### Pros:

* Open source therefore no fees, even for commercial use.
* ACID transaction guarantees
* Built-in support for JSON (is object-relational)
* Many interfaces and predefined functions are available

### Cons:

* Configuration of advanced features, and database setup is difficult to those not well versed in DBMS.
* Slower in read-only operations compared to other RDBMS.

### Ideal Use-case:

* When millions of transactions in distributed environments are possible
* When ACID transaction guarantees are desired.
* When scalability is desired

## MongoDB

The most popular NoSQL document database system.

### Pros:

* Fast performance as relational factors are not required
* Supports JSON natively
* Any data structure can be stored and accessed
* Open-source therefore no commercial fees

### Cons:

* Does not use SQL as its language (uses MongoDB queries).
* Lengthy to setup if not well versed.
* Security features must be manually setup.

### Ideal Use-case:

* When data does not require relationships
* When data is unstructured, or semi-structured
* When handling document data, especially with scalability.

## IBM DB2

One of the first proprietary DBMS, released later as a relational DBMS, with support for Linux, windows and UNIX, systems. Designed for use on IBM servers, but works in standard servers and workstations.

### Pros:

* Native AI support
* Can be hosted on a cloud server or a physical server, or both simultaneously.
* Multi-tasking support, and multi-modal data type support

### Cons:

* Very high cost (fees) for organisations and individuals after 3 years of use.
* Some features such as database clusters require 3rd party additional software.

### Ideal Use-case:

* For large organisations managing and processing large amounts of data, with AI possibilities in data processing.
* For data warehousing
* When ACID transaction guarantees are required.

## Redis

NoSQL support, in-memory database system. Uses a key-value structure for data storage. Suitable for real-time uses such as tracking stock.

### Pros:

* Open source therefore no fees
* Very fast operations due to lack of relational overhead, and in-memory functionality.
* Wide range of data structures (bitmaps, geospatial data, sets, sorted sets, etc.)

### Cons:

* Not much support for relational data storage
* High RAM usage as data resides in RAM for caching
* Restricted to commands only (No query language)

### Ideal Use-case:

* When the database will be responsible for storing a dataset of unstructured data, and does not necessarily require non-volatility
* When data will be accessed frequently (caching performance)

## SQLite

Lightweight, serverless database system, compatible with most platforms available.

### Pros:

* Simple to setup and understand
* Open source therefore no fees.
* Reliable in low-traffic use cases

### Cons:

* Limited throughput in concurrent operations
* Limited overall scalability for maintaining large amounts, and various types of data
* Limited functionality in general, especially in cloud storage (is not a server).

### Ideal Use-case:

* For low traffic web applications, or local applications
* When ACID transaction guarantees are desired.
* For local database systems, especially for personal use cases.

Database Comparison Table (Key-points):Points derived and summarised from [1]-[5]:

| Database | Pros | Cons | Ideal Use-case | Personal factors |
| --- | --- | --- | --- | --- |
| Oracle | - Highly scalable  - Multimodal data-types | - Expensive commercial licensing  -Resource intensive | - Very large databases  - Data warehousing | Unfamiliar |
| MySQL | - Native security  - Prioritising speed over features | -Paid license for enterprise edition  - Difficult to upscale  - Not natively ACID compliant | - For databases focusing on security and speed over features  - Small/Medium web-solutions | Familiar with simple implementations |
| MS SQL Server | - Scalable cloud support  - Dynamic Data masking  - Multi-modal data support | - Expensive commercial licensing  - Resource intensive | - For Microsoft environments  - To shift databases to the cloud | Unfamiliar |
| PostgreSQL | - Open source, no commercial fees  - Highly scalable, with many interfaces | - Difficult to configure advanced features  - Lacking speed in read-only operations | - For high frequency operations, and distributed databases  - Scalability over speed | Unfamiliar |
| MongoDB | - Very fast operations  - No set structure required  - Open source, no commercial fees. | - No support for relational storage  - Does not use SQL  - Requires extensive setup | - For unstructured document storage  -When data does not require relationships | Unfamiliar |
| IBM DB2 | - Native AI support  - Cloud, server, physical support  - Multi-tasking support | - Very expensive fees for all  - Database clusters require 3rd party software | - For large data-heavy databases requiring AI processing  - Data warehousing | Unfamiliar |
| Redis | - in-memory caching support  - Very fast operations  - Open source, no fees | - Little relational storage support  - Restricted to commands only (no query lang.)  - High RAM usage | - When data does not require relationships  - When data will be frequently accessed | Unfamiliar |
| SQLite | - Simple to set up.  - Reliable for low-traffic websites | - Limited throughput  - Limited scalability  - Server-less | - For low traffic web-applications  - For local or personal applications | Familiar with simple implementations |

Difficulty and Licensing derived from [4] and [5]:

| Database | Difficulty | License |
| --- | --- | --- |
| Oracle | Hard learning curve. | Proprietary |
| MySQL | Mild learning curve | Open source, but under GNU: has paid editions for more features. |
| MS SQL Server | Hard learning curve. | Proprietary |
| PostgreSQL | Hard learning curve. | Open-Source |
| MongoDB | Mild learning curve | Open-Source |
| IBM DB2 | Hard learning curve. | Proprietary |
| Redis | Mild learning curve | Open-Source |
| SQLite | Small learning curve | Open-Source |

# Selection of the Top 3 Most Suitable Database Systems

Three database systems which seem to suit the requirements of the Virtual Jukebox application include *MySQL*, *Redis*, and *PostgreSQL*. As core query functionality and security is desirable, and the scope of the application is not substantial or focused around large-scale analytics, simpler databases are preferred over complex, feature-rich, proprietary ones. Oracle, MS SQL Server, and IBM DB2 were ruled out due to their difficult learning curve and their overwhelming feature-set [1]-[4]. SQLite was ruled out due to its lack of features, hindering any form of expandability, as well as its embedded nature which is not relevant in a web-server setting [5]. Relational databases seem more suitable in this scenario, as little to no unstructured data will be handled, therefore MongoDB was ruled out. Redis is a consideration however, due to its ability to be used alongside other database systems as a caching tool to improve performance to frequently accessed data, even though it is a NoSQL system [3].

Redis could be used alongside a traditional RDBMS, simply for caching frequently accessed data such as chat-messages, session user-count, guest-credit count, and the songs in queue. PostgreSQL is another consideration as although it is primarily an RDBMS, it offers support for JSON objects due to its object-relational nature. This would be helpful in the future, if the program would ever need to store unstructured data, without the need for a NoSQL implementation, while still providing all the RDBMS features including query access. MySQL is another option to consider due to its simplicity in implementation, with enough features to suit the Virtual Jukebox’s data storage requirements. Each of these three systems are open-source, allowing for commercial usage without fees if needed [4]. They are also all relatively low in-terms of resource consumption, except for Redis which utilises random access memory more than the other two to obtain caching capabilities [4].

## SQL vs NoSQL Database Systems

An RDBMS is preferred as most if not all data to be stored will follow a specific format, hence will be structured. Structured data is suitable for queries, which would be useful for application functionality such as returning all active jukebox locations, instead of having to extract unstructured data, then search for specific keywords manually. Although unstructured data support would support increased scalability (by not needing to comply with a defined schema), it is not required for the current specification of the software. It is always possible to introduce a NoSQL solution later on for any unstructured data storage requirement, whereas starting with a NoSQL system such as MongoDB would restrict or minimise the ability to query data initially. Additionally, many SQL systems contain support for NoSQL functionality such as JSON data storage in both MySQL and PostgreSQL, which could be used in future scalability if needed [6].

Coloured Comparison Table between Top 3 selections





|  | **MySQL** | **Redis** | **PostgreSQL** |
| --- | --- | --- | --- |
| **Utility** |  |  |  |
| **Performance** |  |  |  |
| **Scalability** |  |  |  |
| **Security** |  |  |  |
| **Difficulty** |  |  |  |
| **NoSQL Support** |  |  |  |

[Note: these colours correspond to a simplified overview of the   
comparisons made in the table below]

`

Detailed Comparison Table between Top 3 selectionsPoints derived and summarised from [4]-[10]:

| Database | Utility | Performance | Scalability |
| --- | --- | --- | --- |
| MySQL | - SQL RDBMS  - Aims to be fast and reliable with a limited feature-set  - Easy to learn  - Base edition is free. Other editions are paid | - Faster in read-only operations (compared to other SQL DBMS)  - Memory Storage Engine provides frequently used table support  - Query cache for frequently used queries | - Scales vertically (Requires more powerful hardware to upscale)  - Less features to support large-scale operations |
| Redis | - NoSQL in-memory database/caching system  - Volatile (data stored in RAM) with options to enable non-volatility  - Key-value storage architecture  - Native data structures including strings, hashes, lists, sets, bitmaps, geospatial indexes, streams, etc.  - Open-source (BSD License) | - High memory (RAM) usage as it is an in-memory cache.  - Low disk usage if used solely as a volatile cache.  - In-memory cache provides very quick  read/write operations (no disk I/O needed except for non-volatility)  - Very fast reads with keys due to its key-value storage structure - O(1) via hashing. | - Larger databases will require more RAM (Virtual memory not supported).  - Unstructured data support enables ease of addition of new data types/structures.  - Horizontally scalable (more machines and/or more RAM to upscale). |
| PostgreSQL | - SQL RDBMS with object support such as JSON  - Aims to provide large-scale performance and scalability with a larger feature-set  - Integration of functions made in programming languages (Java, C, etc.)  - Different user roles/privileges  - Completely Open-source | **-** Faster with complicated queries on large data sets.  - Faster with read-write operations on large data sets with concurrent support (compared to other SQL DBMS)  - Table Partitioning can enable increased performance to frequently accessed parts of a table | - Scales vertically (Requires more powerful hardware to upscale)  - No restriction on database size  - Multi-Version Concurrency Control to support many concurrent users (via parallel queries). |

| Database | Security | Difficulty | NoSQL Support |
| --- | --- | --- | --- |
| MySQL | - TLS encryption support | - Not case sensitive  - Considered easy to set-up and use  - Well-documented | - Later releases provide some NoSQL capabilities including JSON data management/storage. |
| Redis | - Potential data loss from storage of volatile in-memory data.  - Designed for use in trusted environments  - TLS encryption support | - Considered easy/medium to set-up and use  - Mild learning curve compared to other NoSQL and SQL systems  - Commands are case-insensitive, but keys are case sensitive | - Fully NoSQL without native SQL support.  - Unstructured data support |
| PostgreSQL | - Data encryption algorithms supported: AES, 3DES  - Native SSL encryption support for connections | - Case Sensitive  - Generally considered harder to set-up and use | - Some NoSQL support allows for the storage and management of unstructured data (JSON/key-value). |

# Evaluation of Top 3 Databases and Technology Selection

As an RDBMS is preferred for this software, and persistence is required, Redis is ruled out as a standalone/primary database system, however, it is likely to still be used as a caching system alongside a traditional non-volatile SQL RDBMS to enhance performance, which is noted by [10] to be a common design pattern. As Redis will be used, disk I/O will be reduced when reading frequently used database contents due to its on-memory nature [10]. It is likely that contents that do not need to be frequently accessed or do not need real-time access with the Virtual Jukebox will not be stored in-memory using Redis, but will instead solely be kept in the primary RDBMS. Other contents that would require frequent access however, such as playlist contents, and queue contents/song priorities will likely be stored in the Redis on-memory store, with scheduled writes to the disk for backups/persistence. Redis also natively supports data structures such as sorted sets/lists which would be helpful in tracking queue song orders, and geospatial indexes which would be useful in storing active Jukebox locations [9].

For the RDBMS selection between MySQL and PostgreSQL, although MySQL would be sufficient and is generally considered easier to use according to [5], PostgreSQL is preferred due to a variety of reasons. Firstly, PostgreSQL is completely open-source, therefore all its features are available for any form of usage for free, whereas MySQL has fees for increased functionality. PostgreSQL is also more scalable than MySQL as it does not possess a database size limit, and PostgreSQL natively supports Multi-version concurrency control enabling more efficient concurrent operations [6]. Concurrent operation support would be useful when multiple people are requesting songs and/or changing song priorities. Additionally, PostgreSQL natively complies with ACID (Atomicity, Consistency, Isolation, and Durability) transaction properties (whereas MySQL does not natively), enhancing its reliability [6]. Lastly, PostgreSQL supports a range of data encryption algorithms such as AES and 3DES, as well as providing varied NoSQL support in key-value storage such as JSON indexing, and XML support, enabling storage using both dynamic and static schemas, whereas MySQL simply has JSON storage support [11]. This may be useful when upscaling the Virtual Jukebox to include additional features in the future which may require unstructured data storage.

Despite PostgreSQL being more suitable in the specified areas, each of these three database systems will be prototyped with simple databases testing basic functionality to gain a more thorough understanding, and to make a more informed final decision of which system(s) will be used in development. As the difficulties vary between MySQL and PostgreSQL, the prototypes will assist in identifying the feasibility in implementing PostgreSQL versus MySQL.

# References

[1]C. Arsenault. “The Pros and Cons of 8 Popular Databases.” Keycdn.com. https://www.keycdn.com/blog/popular-databases (accessed Mar. 23, 2022).

[2]Idslogic. “The Pros and Cons of Some of the Most Popular Databases.” Idslogic.com. https://www.idslogic.com/blog/2019/09/the-pros-and-cons-of-some-of-the-most-popular-databases.html (accessed Mar. 23, 2022).

[3] M. Kamaruzzaman. “Top 10 Databases to Use in 2021.” TowardsDataScience.com. https://towardsdatascience.com/top-10-databases-to-use-in-2021-d7e6a85402ba (accessed Mar. 23, 2022).

[4] Altexsoft. “Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB. Elasticsearch, and others.” Altexsoft.com. https://www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/ (accessed Mar. 23, 2022).

[5] A. Duggal. “SQLite vs MySQL: 5 Critical Differences.” Hevodata.com. https://hevodata.com/learn/sqlite-vs-mysql/ (accessed Mar. 23, 2022).

[6] A. Phajdar. “PostgreSQL vs MySQL: 8 Critical Differences.” Hevodata.com. https://hevodata.com/learn/postgresql-vs-mysql/ (accessed Mar. 23, 2022).

[7] S. Kakarla. “PostgreSQL vs MySQL.” Sumologic.com. https://www.sumologic.com/blog/postgresql-vs-mysql/ (accessed Mar. 23, 2022).

[8] Redis. “Redis Security.” Redis.io. https://redis.io/docs/manual/security/ (accessed Mar. 27, 2022).

[9] Redis. “Data Structures.” Redis.com. https://redis.com/redis-enterprise/data-structures/ (accessed Mar. 27, 2022).

[10] Redis. “Redis FAQ.” Redis.io. https://redis.io/docs/getting-started/faq/ (accessed Mar. 27, 2022).

[11] 2ndQuadrant. “PostgreSQL vs MySQL.” 2ndQuadrant.com. https://www.2ndquadrant.com/en/postgresql/postgresql-vs-mysql/   
(accessed Mar. 29, 2022).