RESEARCH ASSIGNMENT

June 2021

CMI7509-2021

Carl Wilson [U0370630]

u0370630@unimail.hud.ac.uk

1. RELATIONAL DATABASE: POSTGRES

Relational database management systems [DBMS] are best suited to applications where data can be considered “highly structured” or “structured” such that it fits neatly into a table, known as a relations with a format of columns, known as attributes and rows, known as tuples (Connolly & Begg, 2015). These relations then form said database. As relational databases are quire rigid, they excel where data is predictable and homogeneous – where the same attributes are going to be provided for each tuple.

The rigidity of the relational model means you need to firmly understand the data you are planning to store, it is critical that this schema be laid out prior to creating the database as changes are very difficult afterwards (Harrington, 2016). There are rules that need to be satisfied such as ensuring each tuple is unique within a relation.

A good use case for the relational database would be a hotel room booking system for a hotel chain. Such a system could have relations for customers, rooms, and bookings. For instance, customers would all have: a title, name, address, and contact number. Rooms would have a room number, which could also be a primary key for the relation, a room type (double, twin, family), number of people it could sleep and whether it was smoking or non-smoking for example. Finally, a booking relation would have a customer and room number, which could be foreign keys, number of guests, arrival, and departure date as well as the room rate and whether the room had been pre-paid. The booking relation would then have its own primary key such as a unique booking identifier.

Feature	Type	Remark
customer_id	SMALLINT NOT NULL UNIQUE	Primary Key
title	VARCHAR(6)	
first_name	VARCHAR(32)	
last_name	VARCHAR(32)	
dob	DATE	
street_address	VARCHAR(255)	
city	VARCHAR(32)	
county	VARCHAR(32)	
post_code	VARCHAR(8)	
contact_phone	VARCHAR(16)	
email	VARCHAR(255)	

Feature	Type	Remark
room_number	SMALLINT NOT NULL	Primary Key
type	VARCHAR(16)	
sleeps	SMALLINT	
smoking	SMALLINT	Boolean
disabled_access	SMALLINT	Boolean

Feature	Type	Remark
booking_id	SMALLINT NOT NULL	Primary Key
customer_id	SMALLINT NOT NULL	Foreign Key
room_number	SMALLINT NOT NULL	Foreign Key
no_guests	SMALLINT	
arrival_date	DATE	
depature_date	DATE	
room_rate	MONEY	
paid	SMALLINT	Boolean

While it is key that how the data is going to be stored needs to be properly understood, how the data will be used and queried does not. This is the trade off with a relational database: you don’t need to know how you want to query the database. Provided the data is available in the database – the flexibility of the relational database means that there is always a way to get the data out. This strength is through the join method (Perkins et al., 2018) allowing multiple relations to be joined to form new relations, which can also be aggregated in various ways.

The data stored in hotel room booking relations would not only be predictable and consistent but would also be primarily text based, which Postgres is particularly strong for (Perkins et al., 2018). From these relations it would be easy to run queries to generate invoices and check room availability. While there is a low likelihood of lockout, the ACID compliance would help prevent double booking a room for two guests on the same night.

2. DOCUMENT DATABASE: MONGODB

Where relational databases like Postgres are heavily influenced by the design schema and requires the database developers to understand the data to be stored, with little flexibility – MongoDB requires very little understanding at the outset (Perkins et al., 2018). The focus for MongoDB is processing large quantities of heterogenous data, with MongoDB being a particular strong performer for data v*ariety* (Meier & Kaufmann, 2019). While MongDB is schema-less, it still has structure to the data through JSON and key-value pairs can be used for quick retrieval of documents. This makes them ideal for applications that require very quick retrieval and variety within the objects stored. When considering the *three v's* of big data – MongoDB is employed when variety is crucial.

A good use case would be for reviews, or text-based documentation around a dataset of objects such as movies, music, or books – in this instance I will discuss movies with the specific use case of a movie review website.

It would not be difficult to imagine such a database having collections for movies, reviews, users, and cinemas. Within the movies collection you might have fields to capture data on the cast, the languages, and countries the movie was released in and the category the film fits into or is best aligned to. For example:

> Title: Arrival
>
> ***Category: Drama, Sci-fi***
>
> ***Cast: Amy Adams, Jeremy Renner, Forest Whitaker, Michael Stuhlbarg, Tzi Ma***
>
> Length: 116
>
> Year: 2016
>
> Rating PG-13

It is very possible that there would be multiple or a list of values for cast or category. With a relational database this would be very difficult to manage, it would not only require careful consideration up front but also potentially a lot of built-in redundancy. For this reason, a relational database would not be selected as it would not be the most efficient or effective database available. The strength of MongoDB in this instance is that lists, and arrays can be stored against a field or attribute within a document, and they can be of varying length document to document.

Without the requirement for a schema, the database could easily manage new features that might be added to the application – such as a star rating system. Suppose a field was created on documents stored under the reviews collection for "rating" or "number of stars", to provide some quantified aggregation of reviews a year or two after the application went live online. While this would need to be designed in at the start with a relational model, the MongoDB model would comfortably accommodate the change.

There are drawbacks which can be managed at the front end of the application using forms and schema validation, which can be used to test the underlying structure in a database such as MongoDB.

While MongoDB is not specifically a key-value database, such as Redis [discussed in part 3], it is very strong for quickly searching and identifying objects like videos / images and with indexing and MapReduce can provide very fast searches for datasets that are relatively unstructured.

3. KEY-VALUE STORE: REDIS

When it comes to the *three v's* of big data, Redis is employed when velocity of crucial – anywhere that low latency and high volumes of data is being streamed. Use cases are very broad: real-time analytics services, live Internet of Things [IoT] sensor data, gaming leaderboards, chat/messaging and caching for other database models. The specific case of interest here is live sensor data from a high-speed passenger train.

Sensor data from such a vehicle would be used to monitor the motor, bogeys, cabin and brakes in real time to detect faults, to make a differential diagnosis and to even make predictions about potential failure modes [prognosis] that could be both dangerous and expensive.

Such a vehicle could have two thousand sensors, with a proportion of these running at relatively high frequencies [25 kHz] collecting a total of around 100 GB data per hour, with over 2000 such trains running in China as of 2018 this would total around 200 TB of data to be captured and processed per hour (Xu et al., 2018). Each sensor would provide a stream of potentially high frequency data that would not only be examined in real time against a known baseline to detect said fault but also be written to a non-volatile memory [NVM] such as a relational or document DBMS, for further prognostics development.

The Redis *Sorted Set* can use a time stamp to maintain the order of events the timestamp can be stored as a long integer based on Unix time in nanoseconds [to account for the high frequency] and then used to sort on.

A key for each sensor could be used with a timestamp and event ID, to sort in real-time. In this case an *event* ID has been used rather than *device* ID to capture a unique ID for each transaction for a historical view.

A hash of event ID, timestamp, device ID and *value* would also be captured. The value would depend on the specific sensor i.e temperature, pressure, acceleration, load, or strain. Using the sorted set and hash together would provide both the sorted time of events as well as the details of the event.

A relational database, while capable of storing the data if structured correctly, would not be able to handle the velocity of data – the sheer volume of requests; hence a relational DBMS would not be suitable for this specific application.

4. FINAL COMMENT: POLYGLOT

The reality is none of these databases would be use in isolation, they would almost certainly all be use as polyglot for example: Redis would never be used stand-alone with work be paired a relational or document DBMS.

5. REFERENCES

Connolly, T. M., & Begg, C. E. (2015). *Database systems: a practical approach to design, implementation, and management* (Sixth;Global; ed.). Pearson Education. http://hud.summon.serialssolutions.com/2.0.0/link/0/eLvHCXMwjV3da8IwED_UvUwY7hPdh5TB9jSHTWPaPBarVtDuwcoeQ2ZTBpMKVrd_f5fWDDcY-JIQSMKlTe7ul7vcATjkudv5wxMSwhNuS63cOwnFCqWMzZibUpJ2udQG3oBHUz7xafxaAZM9VTtZfi4Ls2HJtrHxpZ2ly2ioRYapdeHyq-2p4VYH3cjTZfJoAJKtkzExvW2riMu6eHKb_mTqi3nkI2GjaBCI8GUSjKPRzOxColUcTnv6FZgZzXfBoX5mq0Nd5h_IhpBFbXJs5Sv5vk0oxWJPPg1P4UjpRwtnUFHZOTRMqgZrd3IvoBHIjdQSyypDN-eX8DQcxP2wszel2N3lCFSPEDQhpit0e0Js5wpq2SpTTbBcBCPpgrnJm_KorVKPKc_jHlukjElFWQvu92gW5ruKXws7oBPlLXg4iLjrA_vdwDHqIb3yZuMWapv1Vt1BFce04YTScB6IcTSLRd-P2___wG_EGq6J

Harrington, J. L. (2016). *Relational database design and implementation* (4th ed.). Morgan Kaufmann. http://hud.summon.serialssolutions.com/2.0.0/link/0/eLvHCXMwjV1La4NAEB4Sc2mg9E3feChe2gSzbmIslBLyIIc2vVh6lFXXVkgUkhroX-yv6oy4NUgLuYiP0YEZ3HnsNzMAFmubrcqaEIWR7DFbor3tRARwCvoYd5hhj4DkgU8bvCNn9uw8Dbj7VoNvVRrzkYVtMV-IP2odqGqVUI5fas7UY1xsr1OWifPp68jDANj9R8O5MjGofoip5Sm61Rbv5w4zY52731zheGas1kFON3RfjKwdvyfpUtIN6eXsiepzmUljjTxKrvfqV69Dg9FkHg12y6fecOCWmSAMwsyumff7QWPhUD-VojWQuraa0FylAqXBOR42LOBkHxqSyiIOoCaTQ9hTwyD0Ym04gpaC1Ym5TrBTMo96mONDdJGEerxQSHUiOobbydgdTlsb_LwileRVBWWdgJakiTwF3edWwH3bEkLiadd3AnTVBOehY0vW60dncLPNF8-3I7uAHfRcilzIJWikgSuo4yvXVTn_AM7dvzM

Meier, A., & Kaufmann, M. (2019). *SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management* (1st 2019. ed.). Springer. http://hud.summon.serialssolutions.com/2.0.0/link/0/eLvHCXMwjV3fa8IwED7UwZgwtjmH7oeUwXyaI6bplj52Vq1QO4aVPYbWVAYTBavbv79LazYdDHxpKU3aXEvuvrtcvgMw6QNp_dEJDA3Rk83ReHGpSjtwEksqFV1cRMhUqgVe1w6Gtu-w8K0AunqqSrL8nGXLhrnaxosvlSyds6FmFaaWWcqvWk_11op0I53OZFM7SCYaWMrQB-JF9MsIztya4w8dMQ4cHFg_6LrCe_HdQdAf_URmKEVYTy21EUT3pjlVz-_TylCO0g9UQ6iiVukOOi2ni-h9LRnDw5ap6p3CQaL2L5xBIZlX4FAntlfgRBdwMDbz-Ryqo1ffaBrBQp3daBUpm5ZW4b7XDTtea-sVYhPmEYicFJc5Aj6SEXu1zQsozRfzpAZGbMmJhUDNROvDEsJiOolNaj5ybpFI8qgOt1viCP3JxY7MdTC0lCK7v0kfFd3njoVuHKXY5G6voV3u2e4KjhCg2HnI4xpKq-U6uYEi9mnAMWPe2BWDYBSKjhM2_v-z33V_tT8

Perkins, L., Redmond, E., & Wilson, J. R. (2018). *Seven databases in seven weeks : a guide to modern databases and the NoSQL movement* (Second edition. ed.) [still image]. The Pragmatic Bookshelf.

Xu, Q., Zhang, P., Liu, W., Liu, Q., Liu, C., Wang, L., Toprac, A., & Joe Qin, S. (2018). A Platform for Fault Diagnosis of High-Speed Train based on Big Data. *IFAC-PapersOnLine, 51*(18), 309-314. https://doi.org/10.1016/j.ifacol.2018.09.318