Indonesian Journal of Electrical Engineering and Computer Science
Vol. 28, No. 2, November 2022, pp. 1128~1138
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v28.i2.pp1128-1138

Data storage model in low-cost mobile applications

I Made Sukarsa[1], I Kadek Ari Melinia Antara[1], Putu Wira Buana[1], I Putu Agung Bayupati[1], Ni Wayan Wisswani[2], Dina Wahyuni Puteri[1]
[1]Department of Information Technology, Udayana University, Bali, Indonesia
[2]Department of Informatic Management, Bali State Polytechnic, Bali, Indonesia

Article Info

Article history:

Received Feb 10, 2022
Revised Aug 26, 2022
Accepted Sep 6, 2022

Keywords:

Cloud firestore
Database
Hosting
Low cost
NoSQL
SQLite

ABSTRACT

Mobile applications that have data transactions between users require a database relational database management system (RDBMS) and RESTful API operating on the hosting service so that all users can access the data. Renting a hosting service is not cheap and creating a RESTful API takes plenty of time. As an alternative to hosting, a free version of the Cloud Firestore service gives full access rights to the database and has an application programming interface (API) to manage data or access data. However, the free version of Cloud Firestore has limitations in terms of storage capacity, read, write, and delete processes. Therefore, redesigning process of the database was carried out into a low-cost version of the database model consisting of SQLite database and a low-cost version of the NoSQL database to overcome this problem. The goal is to reduce storage space usage and read, write, and delete processing on Cloud Firestore. The low-cost version of the database was tested with 6,030 data. The results obtained were savings of 47.27% storage usage, 83.08% write usage, 91.26% read process usage, and 83.19% delete process usage compared to the test results of the relational database model.

Corresponding Author:

I Made Sukarsa
Department of Information Technology, Udayana University
UNUD Campus Road, Jimbaran, Badung, Bali, Indonesia
Email: sukarsa@unud.ac.id

1. INTRODUCTION

Mobile applications are software that allows for mobility using mobile devices [1]. Mobile applications require data transactions to a relational database management system (RDBMS) via a RESTful application programming interface (API). A good application design must pay attention to database and API design so that optimally synchronization and integration can be carried out [2]. Efficiency in data processing is a challenge in application development, synchronization, integration, and concurrency, which is strongly influenced by processing activities (*read, write, delete*) [3], [4]. The goal is to produce a light/fast, and efficient application in database usage.

Database creation and Restful API are key in the data management process in an application [5]. Hosting services are required so all application users can access that data. Creating or renting a hosting service requires a fee that is not cheap unless you use the free version of the hosting service. Generally, the free version of the hosting service has several drawbacks; it does not provide full access rights to the database, and the server used has the potential to collapse [6].

Application programming interface (API) is a software interface used to facilitate the data exchange between two or more software applications [7]. The RESTful API implements the representational state transfer (REST) architecture to develop web services [8]. According to its function, RESTful APIs are often called RESTful web services or REST web services. The REST architecture is run via hypertext transfer protocol (HTTP) and reads Extensible Markup Language (XML) or JavaScript Object Notation (JSON) files on web pages [9], [10]. The performance of REST web services has been studied to work efficiently both on local services and cloud servers, especially on mobile devices [11]. RESTful APIs can be built using various frameworks and programming languages, where the implementation process takes a long time depending on the data transaction processes that occur in an application [12].

The database becomes a data storage container in making mobile and web applications. The popular database technology used nowadays is the relational database management system (RDBMS). RDBMS has structured data in tables (rows and columns) and has relations between tables connected through primary and foreign keys [13]. RDBMS is the right choice when the type of data used is structured, but if the kind of data used is unstructured and requires high response and speed, the solution that can be used is the NoSQL database [14].

Not only SQL (NoSQL) is a database system that does not have to use structured query language (SQL) commands to perform the data manipulation process [15]. NoSQL, in its practice, is an efficient choice for simplicity, high work analytics, distributed scalability, and good adaptability, which certainly makes the process of storing and retrieving data easier [15]. Furthermore, the performance of query execution speed and the use of NoSQL database storage using MongoDB and Redis has been researched to be better than RDBMS with the percentage of processing time in the range of nanoseconds or milliseconds [16], [17]. RDBMS and NoSQL have their respective advantages based on the type of data that needs to be used. Combining SQL and NoSQL databases can produce more flexible and scalable database management because NoSQL can maximize large amounts of data processing more effectively [18].

Mobile application development includes various aspects in the implementation process, which will undoubtedly require no small expenditure if calculated in terms of costs [19]. Thus, the main focus of this research is minimizing the costs incurred in the application development process but still focusing on the efficiency of memory usage and data processing. One of the Firebase services, namely the free version of Cloud Firestore, can solve development costs, memory, and processing time efficiency problems.

Cloud Firestore is specifically reviewed as being able to be used for non-relational database implementations on mobile devices because it supports mobile client implementations while also can make integration into hosted databases relatively easier [20]. The Cloud Firestore service provides full access rights to the NoSQL database. It has an API to manage data to facilitate data storage, synchronization, and querying data on mobile applications [21]. Cloud Firestore uses a NoSQL database by storing data in collections containing a collection of documents containing data containing keys and values [22]. As a result, cloud Firestore has further complete and faster query features than realtime database services [23]. In addition to the advantages, the free version of Cloud Firestore also has limitations in storage capacity, read, write, and delete processes. Referring to the conditions provided by the Cloud Firestore website as of early November 2021, the storage capacity is only 1 GB, the read process is limited to 50,000 requests per day, the write process is limited to 20,000 requests per day, and the delete process is limited to 20,000 requests per day. If the process or storage area exceeds the usage limit that has been set, it will be charged according to the provisions of the Cloud Firestore service [24]. Thus, these problems can be overcome by redesigning the relational database model into a low-cost version of the database model.

The redesign process in this research uses a low-cost version of the NoSQL database and the SQLite database. SQLite database is used as a data storage medium that can operate locally [25]. The purpose of creating a low-cost version of the database model is to reduce the use of storage and processing (*read*, *write* and *delete*) on the Cloud Firestore service; thereby, it can save on data storage and processing costs. Denormalization will be involved in the migration process from RDBMS to NoSQL, followed by Optimization to get an optimal database design on Cloud Firestore [26]. The optimization process is carried out to eliminate redundancy and ambiguity in the data caused by the denormalization process. In principle, there is no specific method for denormalizing [27].

2. RESEARCH METHODS

The research was conducted by creating a low-cost version of the database model obtained from redesigning the relational database model (RDBMS). The research flow started from the creation of the SQLite database by determining data that can operate locally and not be used for data transactions between users. In contrast, other data or tables will be converted into a low-cost version of the NoSQL database model. These two databases will be the low-cost version of the database model. The research continued with the testing process of the relational database model and the low-cost version of the database on Cloud Firestore. Furthermore, the test results were analyzed and compared. The research flow in the form of a flowchart can see in Figure 1.

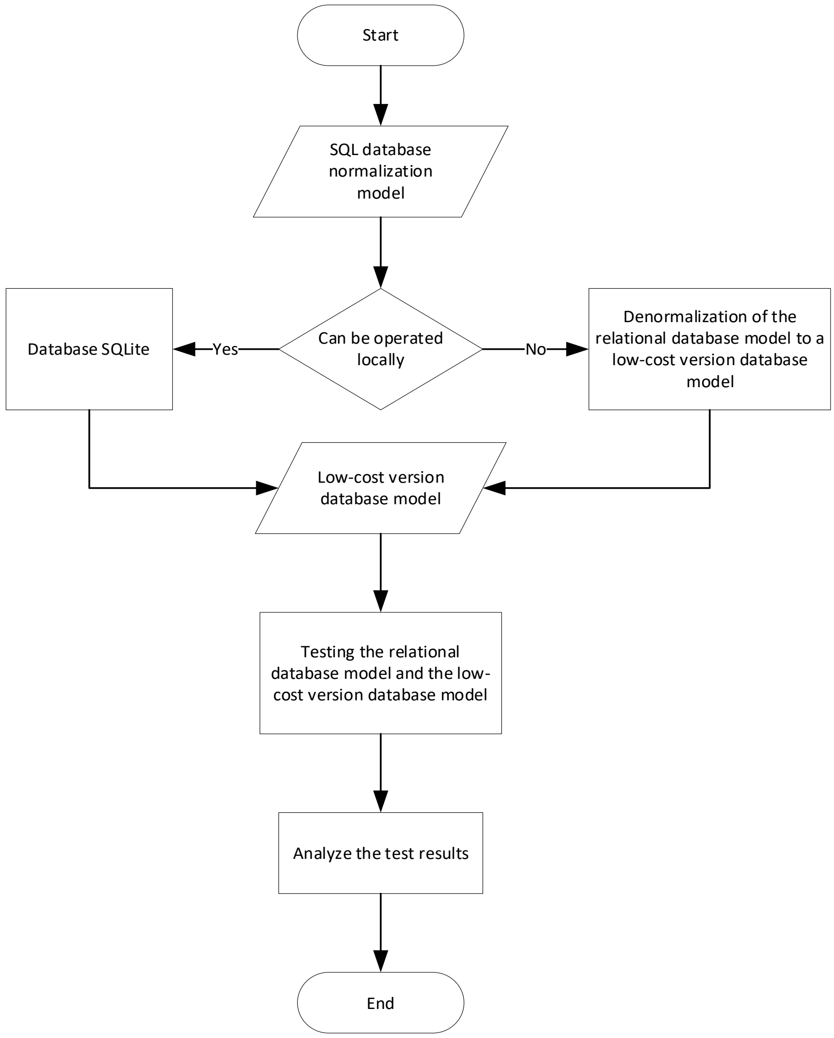


Figure 1. Research flow

2.1. Relational database model

Relational database model is a database structure that has been normalized to a certain level and has a relationship between tables. Normalization is a technique for forming database structures so that most of the ambiguity and data redundancy can be eliminated [28]. The existence of non-constant data will undoubtedly affect the conceptual design of the designed database. Thus the task of normalization becomes very important in the database design process [29]. The relational database created contains five tables related to each other and has their respective functions. User's favorite item data is stored in *tb_favorite*, user data is stored in *tb_user*, item data is stored in *tb_barang*, sales transaction data is stored in *tb_transaksi*, and every detail of the sales amount of goods will be stored in *tb_detail_transaksi*. The relational database design in Figure 2 will redesign to a low-cost version of the database model.

2.2. Redesign process for database optimization

The low-cost version of the database model uses two types of databases, namely SQLite databases to store data that can operate locally and are not used for data transactions between users and a low-cost version of the NoSQL database to be used on Cloud Firestore. SQLite database is an alternative Relational Database Management System that does not require an installation process since it is free and supported by many programming languages [30], [31]. SQLite can define as SQLite database used to store constant data on the final application, where the stored data is data that rarely changes or is static to avoid frequent interactions with the server [32]. Database redesign and optimization done on necessary tables as illustrated in Figure 3. *Tb_favorite* is built on SQLite database because the data in the table can be processed and operated locally and not used for data transactions between users which is illustrated in Figure 3(a). *Tb_transaksi* and *tb_detail_transaksi* are created in the NoSQL database by denormalizing the two tables that have foreign keys. Denormalization is conducted by modifying the table structure and ignoring (controlled) duplicate data to improve database performance [33]. Changes that occur in *tb_transaksi* after denormalization are the column replacement and addition. The *id_user* column is removed and replaced with the name, address, *no_telp*, and email columns. Additional columns are used to store detailed item information in *tb_transaksi*, namely *kode_barang, nama_barang, jumlah* and *harga_jual* as illustrated in Figure 3(b).

Optimization of the data structure is carried out to minimize ambiguity and redundancy in the data. Cloud Firestore has a data type in the form of an array that can be used to store data or transaction detail information without causing redundancy or ambiguity in the data. Fields used to store user information, and item information can be made into two different fields/columns with column names *data_user* and *data_barang* using array data types. Cloud Firestore also has a unique code that is automatically generated for each document to distinguish one document from another. Thus, it can delete the primary key in each table to optimize storage space. The low-cost version of the NoSQL database that will be implemented in Cloud Firestore includes three tables, namely *tb_barang, tb_transaksi*, and User. The data types of the low-cost version of the NoSQL database model have been adapted to the Cloud Firestore service, where the table names in the database model are used as collection names. Further, the column name will be used as the field name in the document where the data is stored, as illustrated in Figure 3(c).

2.3. Test data

The total data used for testing is 6,030 data. The total data consists of 10 item data, 10 user data, 10 favorite data, and 1000 transaction data, with each transaction having five types of goods. Item data will be inputted into the *tb_barang* collection, user data into the *tb_user* collection, favorite data into the *tb_favorite* collection, each of which will be stored in the relational database models and low-cost databases. Overall test data are presented in Tables 1, 2, 3 and 4, 5.

Transaction data will be input into the *tb_transaksi* collection in the relational database model and the low-cost version of the database. The storage process in the low-cost version of the database model is slightly different from the relational database. In the low-cost version of the database model, the user does not have to input *id_user* but instead inputs the name, address, *no_telp*, and email data based on the *id_user*. The transaction detail data that will be input into the *tb_detail_transaksi* collection in the relational database model includes the transaction *id_transaksi, id_barang, jumlah* and *harga_jual barang* fields. Meanwhile, in the low-cost version of the database, the data will be input into the *tb_transaksi* collection in the *data_barang* field, where the field contains *kode_barang, nama_barang, jumlah* and *harga_jual*.

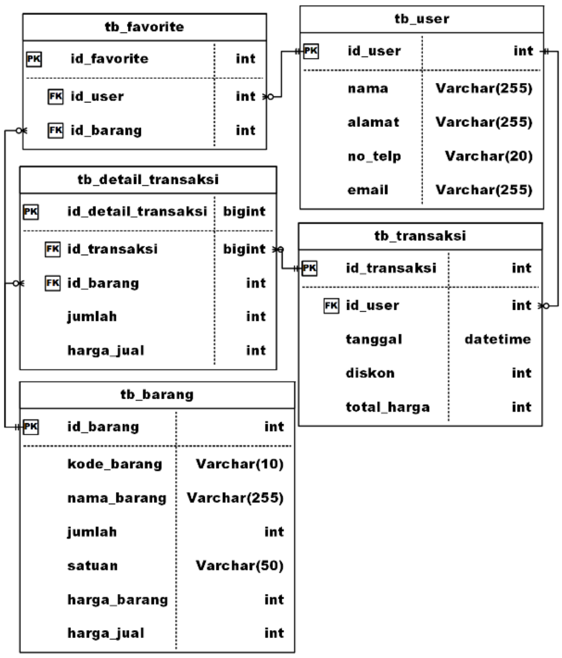
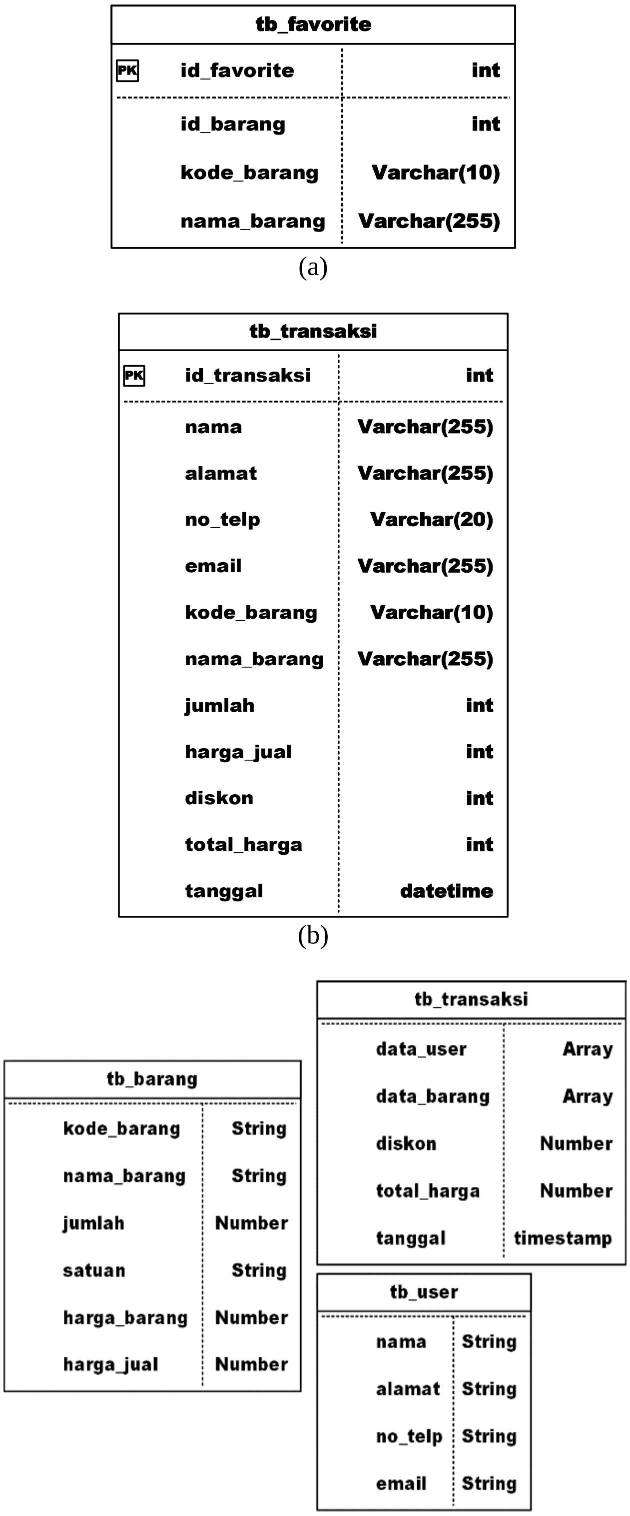


Figure 2. Relational database model



(c)

Figure 3. Database redesign and optimization, (a) *tb_favorite* as SQLite Database, (b) Denormalization of *tb_transaksi* and (c) Low-Cost Version of NoSQL Database Model

Test is done by processing *write* data or inputting item data, user data, favorite data, transaction data, and transaction detail data. The test is continued by reading the transaction data along with user information and item details. Further, do the process of delete user data along with transaction data and details. Tables 1-5 are examples of data in the database structure to be tested.

Table 1. *Tb_barang*

Id	*Kode_brg*	*Nama_brg*	*Jml*	*Satuan*	*Harga_brg*	*Harga_jual*
1	B001	CPU	100	Pcs	3.500.000	4.000.000
2	B002	Monitor	150	Pcs	1.200.000	1.500.000
3	B003	Laptop Asus	200	Pcs	5.700.000	6.500.000
4	B004	Laptop Acer	100	Pcs	6.200.000	7.000.000
5	B005	Mouse	150	Pcs	120.000	150.000
6	B006	Keyboard	150	Pcs	200.000	250.000
7	B007	Printer	50	Pcs	1.250.000	1.750.000
8	B008	Kabel USB	150	Pcs	50.000	75.000
9	B009	Flashdisk	75	Pcs	300.000	350.000
10	B010	HDD	50	Pcs	512.000	600.000

Table 2. *Tb_user*

Id	*Nama*	*Alamat*	*No_telp*	*email*
1	Aprilia	Jln. Mangga 5 blok D 62 RT 001 RW 003 Perumahan Klodran Indah, Klodran, Colomadu, Karanganyar	082122365943	Aprilia@gmail.com
2	Sri Astuti	Jl. Meran No.88 Cilodong	081202365976	Sri.Astuti@gmail.com
3	Annisa	Dk. Ceper RT 01/06, Ds. Ceper, Kec. Ceper, Kab. Klaten	085322365943	Annisa@gmail.com
4	Bella	Blulukan II rt01/06 Colomadu, Karanganyar	087865759393	Bella@gmail.com
5	Dina	Desa Kelet Rt 23 Rw 4 Kecamatan Keling Kabupaten Jepara Provinsi Jawa Tengah	087765570027	Dina@gmail.com
6	Fahdilla	jl tarmidi samarinda kaltim	087765691216	Fahdilla@gmail.com
7	Fitri Ayu	Jajar RT 02/04 Laweyan, Surakarta	081933125331	Fitri.Ayu@gmail.com
8	Putri Ayu	Jalan raya Kedondong desa Cimanuk kecamatan way Lima kab.pesawaran	087864727201	Putri.Ayu@gmail.com
9	Hendra	Banjarsari Nusukan prawit RT 06 RW 03	087864411708	Hendra@gmail.com
10	Indra	Batur citrosono Grabag Magelang Jawa tengah	081907986555	Indra@gmail.com

Table 3. *Tb_favorite*

Id	*Id_user*	*Id_barang*	*Kode_barang*	*Nama_barang*
1	1	1	B001	CPU
2	2	2	B002	Monitor
3	3	3	B003	Laptop Asus
4	4	4	B004	Laptop Acer
5	5	5	B005	Mouse
6	6	6	B006	Keyboard
7	7	7	B007	Printer
8	8	8	B008	USB Cable
9	9	9	B009	Flashdisk
10	10	10	B010	HDD

Table 4. *Tb_transaksi*

Id	*Id_user*	*Tanggal*	*Diskon*	*Total_harga*
1	1	2021-11-04 20:25:40	0	19.150.000
2	2	2021-11-04 19:50:04	0	15.400.000
3	3	2021-11-04 18:00:32	0	15.650.000
4	4	2021-11-04 17:40:53	0	9.225.000
5	5	2021-11-04 16:55:32	0	2.575.000
6	6	2021-11-04 16:04:45	0	3.025.000
7	7	2021-11-04 15:55:44	0	19.150.000
8	8	2021-11-04 15:40:33	0	15.400.000
9	9	2021-11-04 15:05:00	0	15.650.000
10	10	2021-11-04 14:34:05	0	9.225.000
11	1	2021-11-03 13:00:54	0	2.575.000
12	2	2021-11-03 12:04:00	0	3.025.000
dst	dst	Dst	dst	dst
1000	10	2021-07-28 22:11:41	0	9.225.000

Tests are carried out using the Python programming language and the *firebase_admin library*. Python is a programming language that is widely used for the analysis process because it is dynamic, object-oriented, and has good modularity [34]. Python also claims to be a language that combines capabilities, abilities, and an obvious code syntax and is equipped with automatic memory management [35], [36]. Python programming language also has advantages in developing a software product with a large and extensive library [37]. Test conducts for relational databases and a low-cost version in the Python programming language, as illustrated in Figure 4.

Table 5. *Tb_detail_transaksi*

Id	*Id_transaksi*	*Id_barang*	*Jumlah*	*Harga_jual*
1	1	1	1	4.000.000
2	1	2	1	1.500.000
3	1	3	1	6.500.000
4	1	4	1	7.000.000
5	1	5	1	150.000
6	2	2	1	1.500.000
7	2	3	1	6.500.000
8	2	4	1	7.000.000
9	2	5	1	150.000
10	2	6	1	250.000
dst	dst	dst	dst	dst
5000	1000	8	1	75.000

```
Low-Cost Version of Database Testing
Program Code
docUser = db.collection('tb_user')
docBarang = db.collection('tb_barang')
docTrx = db.collection('tb_transaksi')
i = 0
#write data
while i<10 :
   docUser.add({
      'nama': item,
      'alamat': listAlamat[i],
      'no_telp': listNoHp[i],
      'email': listEmail[i],
   })
   docBarang.add({
      'kode_barang':listKodeBrg[i],
 'nama_barang':listNamaBrg[i],
 'jumlah':listJml[i],
 'satuan':'Pcs',
 'harga_barang':listHargaBrg[i],
 'harga_jual':listHargaJual[i],
   })
   i += 1
i = 0
while i < 1000:
   j=0
   dataBarang = []
   while j<5:
      dataBarang.append({

   "kode_barang":listKodeBrg[index],

   "nama_barang":listNamaBrg[index],
         "jumlah":1,
   "harga_jual":listHargaJual[index],
      })
      j+=1
```

```
Low-Cost Version of Database Testing
Program Code
   docTrx.add({
      'data_user':[
         listNama[idUser],
         listAlamat[idUser],
         listNoHp[idUser],
         listEmail[idUser],],
      'data_barang':dataBarang,
      'tanggal':tanggalStr,
      'diskon':0,
      'total_harga': listTotalHarga[i]
   })
#read data
doc_trxData = docTrx.stream()
for doc in doc_trxData:
   print(f'{doc.id} => {doc.to_dict()}')
#delete data
doc_userData = docUser.where('nama', '==',
'Bella').get()
for doc in doc_userData:
   key = doc.id
   docUser.document(key).delete()
   doc_trxData = docTrx.where('data_user',
'array_contains', 'Bella').get()
   for docTrx in doc_trxData:
      keyTrx = docTrx.id
      docTrx.document(keyTrx).delete()
```

Figure 4. Program Code Implementation in *Python*

3. RESULTS AND DISCUSSION

3.1. Test results on the relational database model

The write data process is carried out by inputting user data, item data, favorite data, transaction data, and transaction detail data. The request used for the write data process is 6,030 requests. The use of storage space to accommodate all the data is 0.0011 GB. The read data process is carried out by reading/retrieving transaction data along with user and item information. The request used is 12,000 request read. Finally, the delete process is carried out by deleting user data along with transactions and transaction details. The request used is 601 request read and 601 request delete. The test results are shown in Table 6.

3.2. Test results on the low-cost version of the database model

The write data process is carried out by inputting user data, item data, and transaction data. The request used is 1020 request write. The use of storage space to accommodate all the data is 0.00058 GB. The read data process is carried out by reading/retrieving transaction data along with user and item information. The request used is 1000 requests. The delete process is carried out by deleting user data along with transactions and transaction details. The request used is 101 request read and 101 request delete. The test results are shown in Table 7.

Table 6. Test results on the normalized database model

Testing	Result
Storage Usage	0.0011GB
Write Data	6030 *request write*
Read Data	12000 *request read*
Delete Data	601 *request read* & 601 *request delete*

Table 7. Test results on the low-cost version of the database model

Testing	Result
Storage Usage	0.00058GB
Write Data	1020 *request write*
Read Data	1000 *request read*
Delete Data	101 *request read* & 101 *request delete*

3.3. Test result analysis

The test results show that the use of storage, write processes, read processes, and delete processes in the low-cost version of the database model is smaller than the relational database model. The percentage value of comparison obtained in storage savings of 47.27%, write process of 83.08%, read process of 91.26%, and delete process of 83.19% compared to the relational database model. The percentages generated in Table 8 show that the low-cost version of the database with the Cloud Firestore implementation provides the advantage/savings compared to the relational database. The percentage value generated from the test results obtains using (1).

$$P = (NVH - NN) \div NN\ x\ 100\% \quad (1)$$

The variables description in (1) is as follows. NP is the percentage value wanted to find, and NVH is the value obtained from the test results on the low-cost version of the database model. Besides, NN is the value obtained from the test results on the relational database model. The test results are shown in Table 8.

Table 8. Results comparison

Testing	Model Database		Percentage
	Normalized Version	Low-Cost Version	
Storage Usage	0.0011GB	0.00058GB	47,27%
Write Data	6030 *request write*	1020 *request write*	83,08%
Read Data	12000 *request read*	1000 *request read*	91,26%
Delete Data	601 *request read* & 601 *request delete*	101 *request read* & 101 *request delete*	83,19%

The low cost version model in this study is not only suitable for use on transaction-based (OLTP) as in the example above, but also need to be applied to various services such as data exchange between applications and chatbots [38]-[40].

4. CONCLUSION

The process of redesigning the database to produce a low-cost version of the database model was obtained from breaking the relational database into SQLite and NoSQL databases, followed by a denormalization process. An optimization process will be carried out in the NoSQL database by changing the table structure and data type to become a low-cost version of the NoSQL database. The purpose of making this database model is to save the cost of storing and processing data transactions (write, read and delete) on Cloud Firestore. The database model test was carried out with 6,030 data consisting of 10 item data; 10 user data; 10 favorite data; 1,000 transaction data; and 5,000 transaction detail data. The test results obtained from the low-cost version of the database model were the storage usage of 0.00058GB, the write process usage of 1,020 requests, the read process usage of 1,101 requests, and the delete process usage of 101 requests. In addition, savings in storage usage of 47.27%, the write process usage of 83.08%, the read process usage of 91.26%, and the delete process usage of 83.19% compared to the test results of the relational database model. Furthermore, it is necessary to test database stress with various transactions both in terms of query variations, volume and user redundancy so that a critical point is obtained related to key matters in the application of design in non relational databases.

ACKNOWLEDGEMENTS

Thank you to Indonesia's Ministry of Education, Culture, Research and Technology who has funded this research through the PTUPT scheme with contract number B/107-8/UN14.4.A/PT.01.03/2022. Thanks also to Institute for Research and Community Service (Udayana University) who has facilitated the implementation of this research.

REFERENCES

[1] A. A. G. Singh, E. J. Leavline, and J. Selvam, "Mobile application for m-learning," *International Journal of Advance Research in Computer Science*, vol. 8, no. 3, pp. 313–317, 2017.
[2] R. C. Dinatha, I. M. Sukarsa, and A. A. K. Cahyawan, "Data exchange service using google drive API," *International Journal of Computer Applications*, vol. 154, no. 7, pp. 12–16, 2016.
[3] G. H. Surya, I. M. Sukarsa, and I. G. M. A. Sasmita, "Two-ways database synchronization in homogenous database management system with binary log approach," *Journal of Theoretical and Applied Information Technology*, vol. 65, no. 1, pp. 76–82, 2014.
[4] R. Gudakesa, I. M. Sukarsa, and I. G. M. A. Sasmita, "Two-ways database synchronization in homogeneous DBMS using audit log approach," *Journal of Theoretical and Applied Information Technology*, vol. 65, no. 3, pp. 854–859, 2014.
[5] M. A. Belfedhal and M. Malki, "MASHUP of linked data and Web API," *International Journal of Information Technology and Computer Science*, vol. 10, no. 6, pp. 64–71, 2018, doi: 10.5815/ijitcs.2018.06.07.
[6] C. Louw and C. Nieuwenhuizen, "Digitalization strategies for SMEs: A cost vs. skill approach for website development," *African Journal of Science, Technology, Innovation and Development*, vol. 12, no. 2, pp. 195–202, 2020, doi: 10.1080/20421338.2019.1625591.
[7] A. Agocs and J. M. Le Goff, "A web service based on RESTful API and JSON Schema/JSON Meta Schema to construct knowledge graphs," *CITS 2018 - 2018 International Conference on Computer, Information and Telecommunication Systems*, pp. 1–5, 2018, doi: 10.1109/CITS.2018.8440193.
[8] A. Soni and V. Ranga, "API features individualizing of web services: REST and SOAP," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 9 Special Issue, pp. 664–671, 2019, doi: 10.35940/ijitee.I1107.0789S19.
[9] A. Belkhir, M. Abdellatif, R. Tighilt, N. Moha, Y. G. Gueheneuc, and E. Beaudry, "An observational study on the state of REST API uses in android mobile applications," *Proceedings - 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2019*, pp. 66–75, 2019, doi: 10.1109/MOBILESoft.2019.00020.
[10] D. Rathod, "Performance Evaluation of Restful Web Services and Soap / Wsdl Web Services," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 7, pp. 415–420, 2017, doi: 10.26483/ijarcs.v8i7.4349.
[11] A. Dudhe and S. S. Sherekar, "Performance analysis of SOAP and RESTful Mobile web services in cloud environment," *International Journal of Computer Applications*, pp. 975–8887, 2014.
[12] I M. Sukarsa, I N. Piarsa, and I G. B. P. Putra, "Application of MVP architecture in developing android-based seminar ticket booking applications," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 4, no. 3, pp. 513–520, 2020, doi: 10.29207/resti.v4i3.1396.
[13] T. Aziz, E. Haq, and D. Muhammad, "Performance based comparison between RDBMS and OODBMS," *International Journal of Computer Applications*, vol. 180, no. 17, pp. 42–46, 2018, doi: 10.5120/ijca2018916410.
[14] S. Palanisamy and P. Suvithavani, "A survey on RDBMS and NoSQL Databases MySQL vs MongoDB," *2020 International Conference on Computer Communication and Informatics, ICCCI 2020*, 2020, doi: 10.1109/ICCCI48352.2020.9104047.
[15] S. Venkatraman, K. F. S. Kaspi, and R. Venkatraman, "SQL versus NoSQL movement with big data analytics," *International Journal of Information Technology and Computer Science*, vol. 8, no. 12, pp. 59–66, 2016, doi: 10.5815/ijitcs.2016.12.07.
[16] S. Singh, "Security Analysis of MongoDB," *International Journal for Digital Society*, vol. 10, no. 4, pp. 1556–1561, 2019, doi: 10.20533/ijds.2040.2570.2019.0193.
[17] G. Kaur and J. Kaur, "In-memory data processing using redis database," *International Journal of Computer Applications*, vol. 180, no. 25, pp. 26–31, 2018, doi: 10.5120/ijca2018916589.
[18] M. Potey, M. Digrase, G. Deshmukh, and M. Nerkar, "Database migration from structured database to non- structured database," *International Journal of Computer Applications*, no. Icrtaet, pp. 975–8887, 2015.
[19] I. M. Sukarsa, I. K. G. D. Putra, N. P. Sastra, and L. Jasa, "A new framework for information system development on instant messaging for low cost solution," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 16, no. 6, pp. 2799–2808, 2018, doi: 10.12928/TELKOMNIKA.v16i6.8614.
[20] F. M. Dahunsi, A. J. Joseph, O. A. Sarumi, and O. O. Obe, "Database management system for mobile crowdsourcing applications," *Nigerian Journal of Technology*, vol. 40, no. 4, pp. 713–727, 2021, doi: 10.4314/njt.v40i4.18.
[21] R. B. S. Mathavan, V. Rohitram, C. Ashhwath, and P. Sasikumar, "Drowsiness detection and rest stop suggestion," *Journal of Physics: Conference Series*, vol. 2115, no. 1, p. 012028, 2021, doi: 10.1088/1742-6596/2115/1/012028.
[22] M. Srivastava, V. Yadav, and S. Singh, "Implementation of Web application for disease prediction using AI," *BOHR International Journal of Data Mining and Big Data*, vol. 1, no. 1, pp. 5–9, 2020, doi: 10.54646/bijdmbd.002.

[23] R. A. M. Razid, A. F. Ibrahim, M. N. F. Jamaluddin, and R. A. J. M. Gining, "My-Wakaf: A Waqf of Property Management Application," *Journal of Computing Research and Innovation*, vol. 6, no. 2, pp. 128–141, 2021, doi: 10.24191/jcrinn.v6i2.213.
[24] K.-K. R. Choo, "Mobile Cloud Storage Users," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 20–23, Sep. 2014, doi: 10.1109/MCC.2014.61.
[25] D. Bibicu, L. Moraru, and S. Moldovanu, "Local or external databases in android programming. a practical comparative study," *Annals of Dunarea de Jos University. Fascicle I : Economics and Applied Informatics*, vol. 24, no. 1, pp. 28–32, 2018.
[26] H. J. Kim, E. J. Ko, Y. H. Jeon, and K. H. Lee, "Migration from RDBMS to column-oriented NoSQL: lessons learned and open problems," *Lecture Notes in Electrical Engineering*, vol. 461, pp. 25–33, 2018, doi: 10.1007/978-981-10-6520-0_3.
[27] H. J. Kim, E. J. Ko, Y. H. Jeon, and K. H. Lee, "Techniques and guidelines for effective migration from RDBMS to NoSQL," *Journal of Supercomputing*, vol. 76, no. 10, pp. 7936–7950, 2020, doi: 10.1007/s11227-018-2361-2.
[28] M. S. Vighio, T. J. Khanzada, and M. Kumar, "A tool for query normalization and elimination of redundancy," *Sindh University Research Journal (Science Series)*, vol. 50, pp. 143–147, 2018.
[29] K. Kumar and S. Kumar, "Relational database design: a review," *International Journal of Computer Applications*, vol. 176, no. 6, pp. 14–18, 2017, doi: 10.5120/ijca2017915626.
[30] S. T. Bhosale, T. Patil, and P. Patil, "SQLite: light database system," *International Journal of Computer Science and Mobile Computing*, vol. 44, no. 4, pp. 882–885, 2015.
[31] Y. Wang, Y. Shen, C. Su, J. Ma, L. Liu, and X. Dong, "CryptSQLite: SQLite with high data security," *IEEE Transactions on Computers*, vol. 69, no. 5, pp. 666–678, 2020, doi: 10.1109/TC.2019.2963303.
[32] B. Smitha and K. Shirisha, "Implementation of business register record application on android platform," *International Journal of Computer Applications*, vol. 156, no. 7, pp. 21–26, 2016, doi: 10.5120/ijca2016912464.
[33] N. Kojic and D. Milicev, "Equilibrium of redundancy in relational model for optimized data retrieval," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 9, pp. 1707–1721, 2020, doi: 10.1109/TKDE.2019.2911580.
[34] S. Amghar, S. Cherdal, and S. Mouline, "Storing, preprocessing and analyzing tweets: finding the suitable noSQL system," *International Journal of Computers and Applications*, 2020, doi: 10.1080/1206212X.2020.1846946.
[35] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89497–89509, 2020, doi: 10.1109/ACCESS.2020.2990567.
[36] A. Javed, M. Zaman, M. M. Uddin, and T. Nusrat, "An analysis on python programming language demand and its recent trend in bangladesh," *ACM International Conference Proceeding Series*, pp. 458–465, 2019, doi: 10.1145/3373509.3373540.
[37] K. R. Srinath, "Python -The Fastest Growing Programming Language," *International Research Journal of Engineering and Technology*, vol. 4, no. 12, pp. 354–357, 2017, [Online]. Available: www.irjet.net.
[38] N. W. Wisswani and I. W. K. Wijaya, "Message oriented middleware for library's metadata exchange," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 16, no. 6, p. 2756, Dec. 2018, doi: 10.12928/telkomnika.v16i6.9475.
[39] I. M. Sukarsa, P. W. Buana, and U. Yogantara, "Multi parameter design in AIML framework for balinese calendar knowledge access," *KSII Transactions on Internet and Information Systems*, vol. 14, no. 1, pp. 114–130, Jan. 2020, doi: 10.3837/tiis.2020.01.007.
[40] I. Made Sukarsa, N. W. Wisswani, and P. Wirabuana, "Data exchange delivery between information system at low bandwidth quality using messaging," *Journal of Theoretical and Applied Information Technology*, vol. 60, no. 2, pp. 417–422, 2014.

BIOGRAPHIES OF AUTHORS

I Made Sukarsa he obtained his Doctoral Degree in Udayana University in 2019. He currently works as a lecturer in the Department of Information Technology University of Udayana. His research interests are Natural Language Processing, Integration System, Data Warehouse, Middleware, and Information Technology Governance. Until now, He has written several studies in dozens of international journals indexed scopus. He can be contacted at email: sukarsa@unud.ac.id.

I Kadek Ari Melinia Antara he obtained his Bachelor Degree in Departement of Information Technology at Udayana University, Bali, Indonesia. He has the ability as a web developer, mobile application developer and data management. His research interests are Industry Application. He can be contacted at email: kadek_ari@gmail.com.

Putu Wira Buana he obtained his Master Degree in The Science of Applied Electronics at Brawijaya University in 2007. He currently works as a lecturer in the Department of Information Technology University of Udayana. His research interests are Emerging Technology, And Industry Application. Until now, He has written several studies in dozens of international journals indexed scopus. He can be contacted at email: wbhuana@gmail.com.

I Putu Agung Bayupati received the Bachelor of Engineering degree in Electrical Engineering from Udayana University, and Master of Engineering degree in Information Technology from Bandung Institute of Technology and Ph.D degree in Electrical Engineering and Computer Science from Kanazawa University in 2001, 2006 and 2012 respectively. He joined to Udayana University in 2003 as a lecturer. His research interest are in intelegent signal processsing, computer vision and Business analytics. He can be contacted at email: bayupati@unud.ac.id.

Ni Wayan Wisswani she obtained her Master Degree in Fac. of Electrical Engineering at Udayana University. She currently works as a lecturer in the Department of Informatics Management of Bali. She can be contacted at email: wisswani@pnb.ac.id.

Dina Wahyuni Puteri she graduated from SMAN 1 Melaya in 2018. She is currently in the process of studies at the Department of Information Technology, Udayana University for Bachelors Degree. During in college, she actively participates in student organization, both within in study program, faculty, and university. She can be contacted at email: wahyuni_d@gmail.com.