JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 21 (2021) 378–382
Received: 21 September 2021
Accepted: 18 Ocober 2021

Comparison of Android data storage methods

Porównanie sposobów przechowywania danych w systemie Android

Dominika Kamila Kornaś*

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract

This article presents a comparison of data storage methods available in the Android SDK. Analyzes the following information storage methods: SQLite, Room, Content Providers, SharedPreferences and Preferences DataStore. The aim of the study is to find the relationship between the complexity of the data structure and the cost and efficiency of data storage with the use of given methods. For the purposes of the test, an Android application was created which performed basic data operations and measured their duration. As a result of the performance test, average read and write times for the given data types and sizes were obtained. The summary contains conclusions on the most optimal method of data storage depending on the functional requirements of the application.

Keywords: Android SDK; data storage, mobile device

Streszczenie

Artykuł przedstawia porównanie metod przechowywania danych dostępnych w Android SDK. Poddaje analizie następujące sposoby składowania informacji: SQLite, Room, Dostawcy treści, SharedPreferences oraz Preferences DataStore. Celem przeprowadzonego badania jest znalezienie zależności między złożonością struktury danych, a kosztem i efektywnością ich przechowywania danymi metodami. Na potrzeby testu stworzono aplikację dla systemu Android, która wykonywała podstawowe operacje na danych oraz mierzyła czas ich trwania. W wyniku testu wydajności otrzymano średnie czasy wykonania zapisu i odczytu dla zadanych typów oraz rozmiarów danych. Podsumowanie zawiera wnioski na temat najbardziej optymalnego sposobu przechowywania danych w zależności od wymagań funkcjonalnych aplikacji.

Słowa kluczowe: Android SDK; przechowywanie danych; urządzenie mobilne

*Corresponding author

Email address: **dominika.kornas@outlook.com** (D. K. Kornaś)

1. Wstęp

Nieustanna ewolucja rynku mobilnego stała się przyczyną wykształcenia w społeczeństwie potrzeby ciągłego dostępu do informacji. Urządzenia mobilne będąc w stanie zapewnić dostęp do Internetu, a tym samym prawie nieograniczonej ilości danych, stały się narzędziem dystrybucji informacji. Funkcjonalność dostarczana za pośrednictwem aplikacji mobilnych obecnie znajduje zastosowanie w niemal każdej strefie życia społeczeństwa. Dzięki temu na przestrzeni dekady system Android osiągnął niepodważalny sukces.

Twórcy systemu Android, chcąc sprostać zmieniającym się potrzebom związanym z ciągłym rozwojem sektora aplikacji mobilnych w kolejnych dystrybucjach Android Software Development Kit proponowali udoskonalone lub całkiem nowe metody przechowywania danych. Współcześnie oferowane przez producentów urządzeń mobilnych technologie jedynie przyczyniły się do procesu przemian, pozwalając na wdrażanie wcześniej nieosiągalnych rozwiązań.

Wraz z rosnącą ilością danych, z których zaczęły korzystać aplikacje, poprawa wydajności metod przechowywania danych oraz skrócenie czasu wykonywania z ich użyciem podstawowych operacji było koniecznym dla twórców systemu Android celem, który należało osiągnąć. Jest to związane z bezpośrednim wpływem niniejszych czynników na ogólny odbiór aplikacji przez użytkowników końcowych.

2. Przegląd literatury

Zapewnienie optymalnego czasu operacji na danych stanowi temat rozważań Autorów publikacji „Porównanie możliwości wykorzystania oraz analiza wydajności baz danych na systemach mobilnych”. Badaniu poddano następujące metody przechowywania informacji: SQLite, zewnętrzne serwery bazodanowe (MySQL, PostgreSQL, MicrosoftSQL Server), pliki płaskie, a także dla systemu Android – SharedPreferences, natomiast dla systemu Windows Mobile – Local Settings [1]. W pracy wykazano, że najkrótsze czasy dostępu do danych uzyskał Microsoft SQL Server spośród zewnętrznych serwerów. Dla danych typu prostego najlepszą wydajność osiągnęły ustawienia współdzielone oraz dla danych złożonych – SQLite.

W artykule „AndroBench: Benchmarking the Storage Performance of Android-Based Mobile Devices” przedstawiono wyniki testów wydajnościowych dla operacji zapisu, edycji i usuwania rekordów w bazie danych SQLite [2]. Rezultaty przeprowadzonego badania zostały wyrażone w liczbie transakcji na sekundę (ang. TPS – transactions per second). Czynność wstawiania danych posiada najniższy współczynnik TPS. Między uzyskanymi dla pozostałych operacji wynikami

wystąpiła silna zależność pod względem średnich czasów wykonania.

Autorzy publikacji „Performance analysis on Android SQLite database" zbadali wydajność przetwarzania danych niezaszyfrowanych oraz zaszyfrowanych przez lokalną bazę SQLite [3]. Stwierdzono, że najdłuższy czas wykonania dla rekordów niezaszyfrowanych posiada operacja wstawiania oraz pobierania danych. W przypadku informacji zaszyfrowanych niską efektywnością charakteryzuje się modyfikowanie i usuwanie elementów. Dodatkowo autorzy ustalili, że czas przetwarzania współbieżnego kilku zapytań w porównaniu do czasu niezbędnego na wykonanie pojedynczego zapytania, jest większy średnio o 40 – 44%.

Szczegółowe porównanie rozwiązań odpowiedzialnych za mapowanie obiektowo – relacyjne w systemie Android było przedmiotem publikacji „Porównanie systemów mapowania obiektowo relacyjnego greenDAO
i Room" [4]. Autor wykazał, że Room uzyskuje najkrótsze średnie czasy wykonania zapytań podczas operacji wstawiania danych. Natomiast znacząco mniejszą wydajność osiąga dla czynności aktualizowania i usuwania rekordów. Ponadto korzystanie z biblioteki Room nie wpływa na negatywnie na rozmiar aplikacji w porównaniu do greenDAO, jednakże przyczynia się do zwiększenia użycie pamięci operacyjnej oraz procesora urządzenia. Dodatkowo autor zwrócił uwagę na aspekty związane ze sposobem implementacji omawianych rozwiązań, podkreślając, że użycie Room wymaga podstawowej znajomości języka SQL.

3. Opis obiektu badań

Platforma Android jest przedstawicielem systemów z rodziny UNIX, mających u podstaw jądro Linuxa. Za pośrednictwem interfejsów bazujących na języku Java pozwala komunikować się z systemem operacyjnym
i warstwą sprzętową urządzenia. Zazwyczaj dane można przechowywać w pamięci wewnętrznej lub zewnętrznej urządzenia, przy czym standardowym podejściem jest przechowywanie danych poufnych aplikacji w pamięci wewnętrznej, ze względu na zmniejszone ryzyko utraty dostępu do nich [5 – 7].

Obecnie Android Software Development Kit posiada szereg rozwiązań pozwalających na przetwarzanie i magazynowanie informacji. W tabeli 1 przedstawiono metody przechowywania danych analizowane w ramach niniejszej publikacji.

Tabela 1: Zestawienie metod przechowywania danych

Metoda przechowywania	Typ danych
SQLite	dane ustrukturyzowane
Room	dane ustrukturyzowane
Dostawcy treści	dane ustrukturyzowane
	dane nieustrukturyzowane, tj. pliki audio, wideo, obrazy
Ustawienia współdzielone	pary klucz - wartość
Preferences DataStore	pary klucz - wartość

Biblioteka SQLite znalazła powszechne zastosowanie w aplikacjach dla systemu Android. Jako lekka wersja transakcyjnego silnika bazy danych SQL nie wymaga specjalnej konfiguracji ani serwera, dodatkowo może pracować na urządzeniach o niewielkich zasobach sprzętowych, powodując jedynie niewielkie zużycie pamięci oraz energii. Pliki, zawierające definicję bazy danych, tabel oraz dane są przechowywane w pamięci urządzenia. SQLite wspiera relacyjny model danych i obsługę dynamicznych, złożonych zapytań [8].

Niskopoziomowe operacje konieczne do obsługi bazy danych SQLite wyeliminowano wprowadzając bibliotekę Room, która pozwoliła na wydzielenie warstwy logiki biznesowej oraz bezpośrednie odwzorowanie obiektów do postaci tabel relacyjnej bazy danych za pomocą techniki mapowania obiektowo – relacyjnego. Połączenie z bazą danych, reprezentacja tabel oraz obsługa zapytań jest realizowana za pośrednictwem trzech głównych komponentów, kolejno: *@Database, @Entity* oraz *@Dao*. Ponadto wykorzystanie Room wprowadziło możliwość statycznej analizy kodu oraz weryfikacji poprawności zapytań w czasie kompilacji [9].

Zarówno SQLite, jak i Room, nie pozwalają na dostęp do danych aplikacji zewnętrznym procesom. Możliwość przetwarzania informacji, składowanych przez inne aplikacje na urządzeniu, oferują w systemie Android dostawcy treści. Dane mogą posiadać różną postać, np. tabele bazodanowe, obrazy, pliki audio i wideo oraz pochodzić z kilku źródeł. Jedynym wymogiem jest statyczna, trwała lokalizacja, do której aplikacja będzie miała dostęp, który odbywa się za pośrednictwem schematu URI (Uniform Resource Identifier). Na jego podstawie aplikacja może uzyskać ogólny lub czasowy dostęp do wybranych zasobów. Dostawcy treści stanowili warstwę abstrakcji dla bazy danych SQLite przed wprowadzeniem biblioteki Room. Dodatkowo, jako jedyny
z analizowanych sposobów przechowywania danych, pozwalają na zachowanie danych nawet po odinstalowaniu aplikacji z pamięci urządzenia [10].

Przechowywanie prostych typów danych w postaci klucz – wartość w systemie Android może zostać zrealizowane za pomocą dwóch mechanizmów: ustawień współdzielonych (ang. SharedPreferences) lub wprowadzonego w 2020 roku jako ich następca – Preferences DataStore. Oba rozwiązania dostarczają możliwość wykonania jedynie operacji zapisu i odczytu danych. Operacja modyfikacji polega na ponownym zapisie nowej wartości o tym samym kluczu. Natomiast całkowite usunięcie danych z pamięci urządzenia nie jest możliwe, dopóki aplikacja nie zostanie całkowicie odinstalowana. Ustawienia współdzielone opierają się głównie na synchronicznych operacjach na danych, w przeciwieństwie do Preferences DataStore, który dzięki wykorzystaniu komponentów języka Kotlin, zapewnia ich asynchroniczne wykonanie. Dodatkowo ustawienia współdzielone wspierają dostęp zewnętrznych procesów do składowanych informacji [11 – 12].

4. Metoda badań

Celem badań było przetestowanie wydajności wybranych metod przechowywania danych dostępnych w Android Software Development Kit. Skupiono się na analizie efektywności wykonania operacji wstawiania oraz pobierania informacji, ponieważ jako jedyne są w sposób bezpośredni dostarczane przez wszystkie analizowane technologie.

Do przeprowadzenia testów konieczne było opracowanie struktur danych, które zostaną poddane analizie oraz skonstruowanie generatora, który pozwoli na ich uzyskanie. Rezultatem jego działania była kolekcja elementów w postaci listy o zadanym rozmiarze. Przetestowano wydajność aplikacji podczas przetwarzania 100, 1000, 10000 oraz 50000 rekordów. Natomiast typy i zakresy danych wykorzystane podczas badania znalazły się w tabeli 2.

Tabela 2: Typy i zakresy danych biorące udział w badaniu

Typ danych	Zakres danych
liczba całkowita	32 bity
liczba rzeczywista	64 bity
ciąg znakowy	500 znaków
obiekt	liczba całkowita 32 – bitowa liczba rzeczywista 64 – bitowa ciąg 500 – znakowy pole typu boolean
para klucz – wartość	klucz typu liczbowego wartość w postaci ciągu 100 – znakowego

W przypadku SQLite, Room oraz dostawców treści, będących pośrednikiem dla SQLite, dane zostały odwzorowane na rekordy bazodanowe, np. pola obiektów odtworzono jako odpowiadające im kolumny w tabeli. Metodami, które wymagały parsowania danych okazały się SharedPreferences oraz Preferences DataStore. Jako rezultat przyjęto średnią z uzyskanych podczas 5 pomiarów wyników.

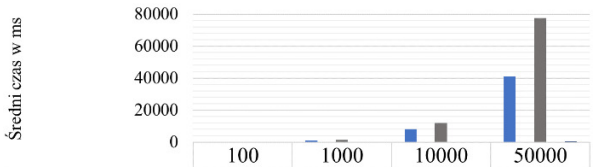
Każdy scenariusz badawczy został zrealizowany na podstawie ogólnego schematu:

1. Wybór parametrów generatora danych pseudolosowych: liczby rekordów, typu i zakresu danych.
2. Generowanie danych.
3. Wybór jednej z operacji: wstawianie lub odczyt rekordów.
4. Pomiar czasu wykonania wybranej operacji dla każdej z analizowanych metod przechowywania danych.

Czas wykonania wybranych operacji był mierzony w każdym przypadku od momentu wysłania żądania o jej przeprowadzeniu do otrzymania odpowiedzi o sukcesie wykonania. Dla operacji wymagających dodatkowego przygotowania danych, czas parsowania był wliczany
do końcowego wyniku.

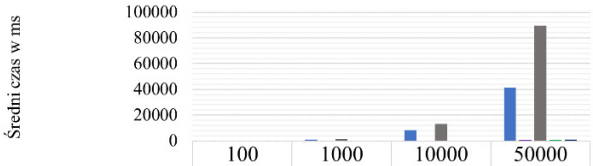
5. Wyniki

Poniżej przedstawiono wyniki przeprowadzonych badań. Wykresy widoczne na rys. 1 do rys. 5 prezentują kolejno rezultaty pomiarów czasu dla operacji wstawiania danych następujących typów: liczby całkowite, liczby rzeczywiste, ciągi znakowe, obiekty oraz pary klucz – wartość. Zaobserwowano, że liczba rekordów wpływa na kolejność analizowanych metod przechowywania danych
pod względem wydajności operacji zapisu. Przypadki testowe, w których przetwarzano 100 oraz 1000 rekordów wykazały, że najlepszy średni czas zapisu osiągał Room dla wszystkich typów danych. Na kolejnym miejscu znalazły się ustawienia współdzielone. Jednakże po zwiększeniu liczby rekordów do 10000 ustawienia współdzielone uzyskały lepszy czas przetwarzania danych niż Room. Jedynym wyjątkiem były liczby rzeczywiste, gdzie Room uzyskał krótsze średnie czasy dla każdej analizowanej liczby rekordów. Podobna sytuacja miała miejsce dla 50000 elementów. Dla wszystkich analizowanych ilości oraz typów danych na kolejnych miejscach znalazły się DataStore, SQLite oraz dostawcy treści. Dodatkowo DataStore nie był w stanie wykonać zapisu 50000 obiektów oraz par klucz – wartość podczas żadnej z wykonanych prób. Tę sytuację zobrazowano na wykresach wynikiem 0 dla tego scenariusza testowego.



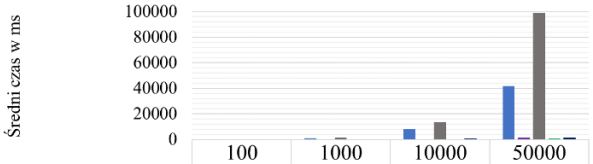
	100	1000	10000	50000
SQLite	122,2	979,2	8270,4	41029,2
Room	2	8,6	49,8	289,2
Dostawcy treści	248,6	1412,2	12080,6	77403
SharedPreferences	13,8	22	29,8	101
DataStore	67,8	96,8	136,8	470,4

Rysunek 1: Porównanie średnich czasów operacji wstawiania danych dla liczb całkowitych w zależności od liczby rekordów.



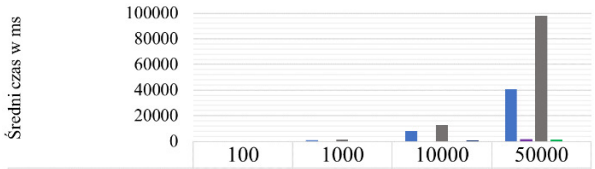
	100	1000	10000	50000
SQLite	127,8	1079,8	8195,4	41248,4
Room	2,8	8	47,2	468
Dostawcy treści	252,4	1521,6	13030,6	89639,2
SharedPreferences	21,6	59,4	97,8	620,2
DataStore	64,4	70,2	234,6	985,2

Rysunek 2: Porównanie średnich czasów operacji wstawiania danych dla liczb rzeczywistych w zależności od liczby rekordów.



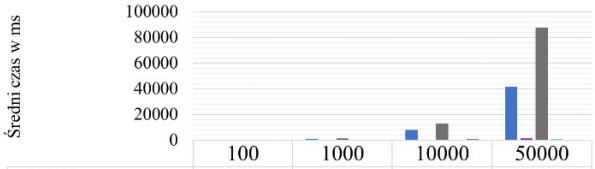
	100	1000	10000	50000
SQLite	137,8	971,2	7997	41654,8
Room	4,6	31,2	262,4	1383,8
Dostawcy treści	248	1486,8	13381,2	99236
SharedPreferences	20,2	39,6	82	469,8
DataStore	82,6	104,2	516,6	1461,2

Rysunek 3: Porównanie średnich czasów operacji wstawiania danych dla ciągów znakowych w zależności od liczby rekordów.



	100	1000	10000	50000
SQLite	117,8	945,8	7953,4	40819,6
Room	6	30,8	323	1560,4
Dostawcy treści	229	1461	12866,8	97816,6
SharedPreferences	27,4	64,4	236,6	1314,4
DataStore	80,4	135,8	667	0

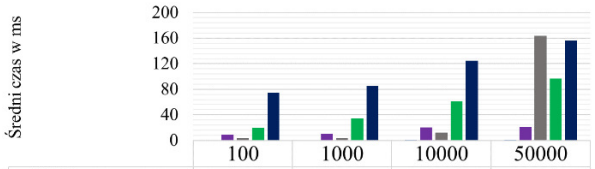
Rysunek 4: Porównanie średnich czasów operacji wstawiania danych dla obiektów w zależności od liczby rekordów.



	100	1000	10000	50000
SQLite	113	853,2	7971,2	41336
Room	4,4	26,6	284,6	1354,4
Dostawcy treści	242,6	1388,8	12601,4	87716,2
SharedPreferences	19,6	59,8	133,2	647,2
DataStore	76,8	163	617,8	0

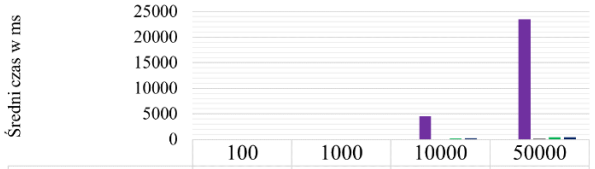
Rysunek 5: Porównanie średnich czasów operacji wstawiania danych dla par klucz – wartość w zależności od liczby rekordów.

Na wykresach (rys. 6 – rys. 10) znalazły się wyniki analizy operacji odczytu. W każdym przypadku testowym najkrótszy średni czas osiąga SQLite, na kolejnym miejscu klasyfikują się dostawcy treści. Dla liczb całkowitych oraz obiektów następne pozycje zajmują Room, SharedPreferences oraz DataStore. Taka sama kolejność zachowana jest dla pozostałych typów danych podczas wykonywania operacji pobierania 100 oraz 1000 rekordów. Natomiast dla 10000 i 50000 elementów Room uzyskuje najdłuższy średni czas.



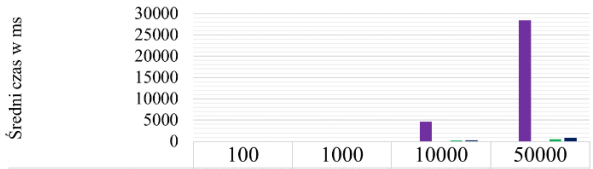
	100	1000	10000	50000
SQLite	0,56	0,75	1,26	1,32
Room	9,6	10	20,2	21,2
Dostawcy treści	3,2	3,4	12,4	164
SharedPreferences	19,6	34	61,8	97
DataStore	74,4	84,8	125	156,4

Rysunek 6: Porównanie średnich czasów operacji odczytywania danych dla liczb całkowitych w zależności od liczby rekordów.



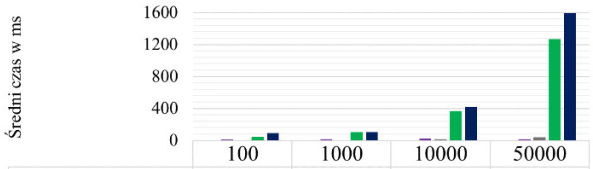
	100	1000	10000	50000
SQLite	0,49	0,55	1,17	1,1
Room	12	15,2	4541,2	23465,8
Dostawcy treści	3,8	4	12,2	164,2
SharedPreferences	28	75	167,2	398,4
DataStore	57,4	85,4	208,8	410,6

Rysunek 7: Porównanie średnich czasów operacji odczytywania danych dla liczb rzeczywistych w zależności od liczby rekordów.



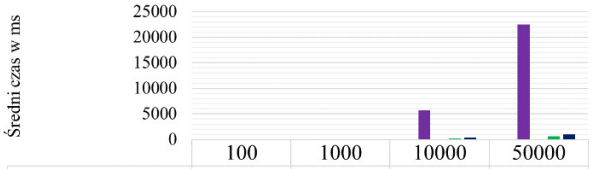
	100	1000	10000	50000
SQLite	0,45	0,56	1,31	1,25
Room	12,6	22,8	4555	28445,2
Dostawcy treści	3,6	6	12,6	31
SharedPreferences	24	41	130,4	491
DataStore	87,6	94,2	263,2	810

Rysunek 8: Porównanie średnich czasów operacji odczytywania danych dla ciągów znakowych w zależności od liczby rekordów.



	100	1000	10000	50000
SQLite	0,48	0,5	1,15	1,37
Room	11	14,2	21,6	18,2
Dostawcy treści	4	5,6	14,8	37,2
SharedPreferences	44	101,8	365,6	1265,6
DataStore	93,2	105,4	421,2	1593,6

Rysunek 9: Porównanie średnich czasów operacji odczytywania danych dla obiektów w zależności od liczby rekordów.



	100	1000	10000	50000
SQLite	0,42	0,68	1,07	0,93
Room	11,2	21	5621,6	22510,8
Dostawcy treści	3,4	4,6	14	34,2
SharedPreferences	25,8	54,2	185,4	628,8
DataStore	77	84,2	315,4	975,8

Rysunek 10: Porównanie średnich czasów operacji odczytywania danych dla par klucz – wartość w zależności od liczby rekordów.

6. Wnioski

Jednoznaczne wskazanie najbardziej uniwersalnej metody przechowywania danych spośród tych, które udostępnia Android SDK nie jest możliwe. Należy wziąć pod uwagę nie tylko kluczowe czynniki, jakie stanowią typ informacji oraz ich ilość, ale także szczegółowe wymagania biznesowe tworzonej aplikacji. Pozwoli to wyeliminowanie najmniej optymalnych dla danego oprogramowania rozwiązań już na etapie projektowania.

W związku z powyższym najbardziej optymalne metody dla aplikacji wykonujących częste operacje wstawiania danych stanowią Room oraz SharedPreferences. Dodatkowo należy zwrócić uwagę, że obecnie promowaną i zalecaną przez dokumentację metodą jest Room, będący jednym z przedstawicieli systemów ORM. Redukuje tym samym liczbę zapytań koniecznych do napisania przez programistę w języku SQL. Ustawienia współdzielone korzystają z danych typu klucz – wartość.

W związku z tym przechowywanie za ich pomocą złożonych struktur informacji wymaga dodatkowego nakładu pracy np. parsowania danych.

DataStore jako jedyna metoda przechowywania danych nie ukończył operacji zapisu wywołując błąd braku pamięci operacyjnej. Związane jest to z koniecznością utworzenia puli wątków asynchronicznych, z których każdy posiada zarezerwowaną ilość zasobów. Ponadto, aby skorzystać z tego rozwiązania w aplikacji tworzonej w języku Java, należy użyć biblioteki RxJava. Tylko w języku Kotlin jest to metoda dostarczana natywnie.

Podczas wykonania operacji odczytywania danych szczególnymi wynikami wykazał się SQLite, uzyskując istotnie krótsze średnie czasy przetwarzania zapytania niezależnie od typu i liczby rekordów. Kolejną pozycję ze względu na otrzymane wyniki zajmują dostawcy treści. Należy jednak zaznaczyć, że niniejsze metody przechowywania danych są jednymi ze starszych rozwiązań, które w nowych aplikacjach najczęściej są zastępowane nowszymi propozycjami.

Uzyskane dla operacji zapisu i odczytu danych wyniki są przeciwne. Metody uzyskujące najlepsze średnie czasy pobierania, otrzymały najgorsze rezultaty podczas wstawiania rekordów. Ostatecznie to do programisty należy wybór rozwiązania, które spełni wymagania funkcjonalne oraz zapewni maksymalne doświadczenie użytkownika podczas korzystania z aplikacji.

Literatura

[1] M. Grudzień, K. Korgol, D. Gutek, Porównanie możliwości wykorzystania oraz analiza wydajności baz danych na systemach mobilnych, Journal of Computer Sciences Institute 2 (2016) 133-139, https://doi.org/10.35784/jcsi.129.

[2] J. Kim, J. Kim, AndroBench: Benchmarking the Storage Performance of Android-Based Mobile Devices, Frontiers in Computer Education (2012) 667-674, https://doi.org/10.1007/978-3-642-27552-4_89.

[3] N. Obradovic, A. Kelec, I. Dujlovic, Performance analysis on Android SQLite database, 2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH) (2019) 1-4, https://ieeexplore.ieee.org/document/8717652.

[4] M. Lewiński, Porównanie systemów mapowania obiektowo relacyjnego greenDao i Room, Journal of Computer Sciences Institute 14 (2020) 43-47, https://doi.org/10.35784/jcsi.1574.

[5] N. Gandhewar, R. Sheikh, Google Android: An Emerging Software Platform For Mobile Devices, International Journal on Computer Science and Engineering (2010) 12–17.

[6] K. Honcharenko, J. Smołka, Analiza rozwoju środowiska uruchomieniowego systemu Android. Journal of Computer Sciences Institute 12 (2019) 246–251, https://doi.org/10.35784/jcsi.504.

[7] Metody przechowywania danych i plików – dokumentacja Android, https://developer.android.com/training/data-storage, [04.05.2021]

[8] S. T. Bhosale, T. Patil, P. Patil, SQLite: Light Database System. International Journal of Computer Science and Mobile Computing 4 (2015) 882–885.

[9] Biblioteka Room – dokumentacja Android, https://developer.android.com/training/data-storage/room, [07.05.2021]

[10] Dostawcy treści – dokumentacja Android, https://developer.android.com/guide/topics/providers/content-providers, [04.05.2021]

[11] Ustawienia współdzielone – dokumentacja Android, https://developer.android.com/training/data-storage/shared-preferences, [07.05.2021]

[12] DataStore – dokumentacja Android, https://developer.android.com/topic/libraries/architecture/datastore, [04.05.2021]