Ветчинов Владислав Олегович
бакалавр, студент
Должиков Валерий Михайлович
бакалавр, студент
Научный руководитель
Лебедев Константин Андреевич
д-р физ.-мат. наук, профессор
ФГБОУ ВО «Кубанский государственный университет»
г. Краснодар, Краснодарский край

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ЛОКАЛЬНЫХ БАЗ ДАННЫХ В МОБИЛЬНЫХ ПРИЛОЖЕНИЯХ НА FLUTTER: SQLITE, HIVE И REALM

Аннотация: *в статье проводится сравнительный анализ трёх подходов к локальному хранению данных в мобильных приложениях, разработанных на фреймворке Flutter: реляционной СУБД SQLite (через ORM-библиотеку Drift), NoSQL-хранилища Hive и объектной базы данных Realm. Сравнение выполнено по критериям, существенным для учебных и прикладных проектов: порог входа, типобезопасность, производительность, поддержка миграций и пригодность для офлайн-сценариев. Практической основой послужил опыт разработки мобильного приложения «Фитнес-трекер», в котором локальная база данных выступает единственным хранилищем. Результаты адресованы преподавателям и студентам, выбирающим инструментарий для учебных проектов по мобильной разработке.*

Ключевые слова: *SQLite, Hive, Realm, Flutter, локальная база данных, мобильная разработка, Drift, офлайн-приложение, сравнительный анализ.*

Мобильное приложение, претендующее на работу без постоянного интернет-подключения, нуждается в надёжном локальном хранилище. Выбор конкретной технологии влияет на архитектуру проекта, скорость разработки и возможности масштабирования. Для экосистемы Flutter наиболее распространены три

варианта: SQLite (обычно через ORM-обёртки sqflite или Drift), Hive и Realm. Каждый из них опирается на различные модели данных и предъявляет разные требования к разработчику [1].

SQLite и библиотека Drift. SQLite – встраиваемая реляционная СУБД, присутствующая в каждом Android- и iOS-устройстве. Она поддерживает стандарт SQL, транзакции ACID, индексы и представления при размере библиотеки около 600 КБ. Во Flutter для работы с SQLite чаще всего используют пакет sqflite, предоставляющий низкоуровневый доступ через SQL-строки, или Drift (ранее Moor) – типобезопасную ORM-библиотеку с кодогенерацией [2].

Drift позволяет описывать таблицы как классы Dart: каждое поле становится колонкой с указанием типа, значения по умолчанию и ограничений. При запуске build_runner генерируются классы-модели, companion-объекты для вставки и типобезопасные выражения для запросов. Ошибки в структуре запросов обнаруживаются на этапе компиляции, а не во время выполнения, что критически важно для студентов, ещё не привыкших к отладке SQL.

В приложении «Фитнес-трекер» Drift обслуживает три таблицы: Users (профиль с десятью полями), DailyMeals (приёмы пищи с привязкой к дате) и DailyStats (ежедневная статистика шагов и калорий). За время разработки схема прошла четыре версии, и каждая миграция – добавление таблицы или колонки – потребовала написания явного кода обновления. Именно механизм миграций стал наиболее трудоёмкой частью работы с Drift: при некорректной миграции приложение не запускается у пользователя, обновившего его с предыдущей версии.

Hive. Hive – легковесное NoSQL-хранилище типа «ключ-значение», написанное целиком на Dart без нативных зависимостей. Данные хранятся в бинарных файлах (boxes), доступ к которым не требует SQL-запросов: достаточно вызвать box.get(key) или box.put(key, value). Hive поддерживает хранение примитивов, списков, карт и пользовательских объектов через адаптеры типов, генерируемые с помощью build_runner [3].

Главное преимущество Hive – простота и скорость. Для учебного проекта, где данные представляют собой плоские записи без сложных связей, Hive позволяет организовать хранение за несколько строк кода. Не нужно описывать таблицы, писать SQL, настраивать миграции – достаточно открыть box и записать объект.

Однако у Hive есть существенные ограничения. Он не поддерживает реляционные связи между сущностями: если приёмы пищи нужно связать с конкретным блюдом из справочника, разработчик вынужден самостоятельно реализовывать логику поиска по идентификатору. Запросы с фильтрацией и сортировкой приходится писать в коде Dart, перебирая все записи в box, что при большом объёме данных приводит к снижению производительности. Кроме того, Hive не имеет встроенного механизма миграций: при изменении структуры объекта разработчик отвечает за совместимость самостоятельно.

Realm. Realm – объектная база данных, изначально разработанная для мобильных платформ и впоследствии приобретённая MongoDB. Для Flutter доступен пакет realm, позволяющий описывать модели данных как классы Dart с аннотациями и работать с ними через объектный API без SQL [4].

Realm сочетает удобство объектного подхода с возможностями, характерными для реляционных СУБД: поддержка связей между объектами (один-к-одному, один-ко-многим), фильтрация через выражения запросов (Realm Query Language), реактивные потоки для отслеживания изменений и встроенная синхронизация с облаком MongoDB Atlas. Для учебных проектов, предполагающих в будущем добавление облачной синхронизации, Realm может оказаться привлекательным выбором.

К недостаткам Realm относятся: значительный размер нативной библиотеки (увеличение APK на 5–10 МБ), менее зрелая поддержка Flutter по сравнению с нативными SDK для Android и iOS, а также более сложная модель потокобезопасности – объекты Realm привязаны к потоку, в котором были созданы, что требует от студента понимания изоляции данных между изолятами (isolates)

Dart. Сравнительный анализ трёх подходов по ключевым критериям представлен в таблице 1.

Таблица 1

Сравнение локальных баз данных для Flutter

Критерий	SQLite (Drift)	Hive	Realm
Модель данных	Реляционная	Ключ-значение	Объектная
Язык запросов	SQL (типобезопасный)	Dart-код	RQL
Типобезопасность	Высокая (кодогенерация)	Средняя (адаптеры)	Высокая (аннотации)
Миграции схемы	Встроенные	Отсутствуют	Автоматические
Реляционные связи	Полные (SQL JOIN)	Нет	Да (объектные)
Реактивные потоки	Да (Stream)	Да (listenable)	Да (Stream)
Размер библиотеки	~600 КБ	~200 КБ	5–10 МБ
Облачная синхронизация	Нет	Нет	Да (Atlas)
Порог входа	Средний	Низкий	Средний-высокий
Пригодность для учебных проектов	Высокая	Высокая (простые)	Средняя

Рекомендации по выбору. Для учебных проектов начального уровня, где данные представляют собой простые записи без связей (заметки, списки дел, настройки), Hive остаётся наиболее быстрым способом организовать локальное хранение: минимальный объём кода, отсутствие SQL и нативных зависимостей.

Для проектов среднего и высокого уровня сложности, где данные связаны между собой и требуется надёжный механизм миграций, предпочтителен SQLite через Drift. Именно этот вариант был выбран для «Фитнес-трекера»: три связанные таблицы, четыре версии схемы, запросы с фильтрацией по дате и типу приёма пищи – задачи, для которых реляционная модель подходит естественным образом [5].

Realm целесообразно рассматривать, когда проект изначально предполагает облачную синхронизацию или когда объектная модель данных значительно удобнее реляционной. Однако увеличение размера APK и необходимость

разбираться с потокобезопасностью делают Realm менее удобным для первых учебных проектов.

Независимо от выбора конкретной технологии, работа с локальными базами данных в рамках учебных проектов формирует у студентов важные компетенции: понимание моделей данных, навыки проектирования схем, опыт работы с миграциями и осознание компромиссов между простотой и функциональностью – знания, применимые далеко за пределами экосистемы Flutter [6].

Заключение.

Проведённый сравнительный анализ трёх подходов к локальному хранению данных в мобильных приложениях на Flutter показал, что каждый из инструментов занимает свою нишу и оптимален для определённого класса задач. Hive обеспечивает минимальный порог входа и подходит для простых учебных проектов с плоской структурой данных. SQLite в связке с библиотекой Drift предоставляет типобезопасный доступ к реляционной модели, встроенные миграции и полноценный язык запросов, что делает его предпочтительным выбором для проектов средней и высокой сложности. Realm предлагает объектную модель данных и встроенную облачную синхронизацию, однако его применение в учебных проектах ограничено увеличенным размером библиотеки и повышенными требованиями к квалификации разработчика.

Опыт разработки приложения «Фитнес-трекер» подтвердил, что для проектов с несколькими связанными сущностями и необходимостью версионирования схемы SQLite через Drift является наиболее сбалансированным решением. Независимо от выбранного инструмента, практическая работа с локальными базами данных формирует у студентов системное понимание моделей данных и архитектурных компромиссов – компетенции, востребованные далеко за пределами экосистемы Flutter.

Список литературы

1. Flutter documentation: Build apps for any screen. URL: https://docs.flutter.dev/ (дата обращения: 10.04.2026).

2. Drift documentation: Reactive persistence library for Flutter and Dart. URL: https://drift.simonbinder.eu/docs/ (дата обращения: 10.04.2026).

3. Hive documentation. URL: https://docs.hivedb.dev/ (дата обращения: 10.04.2026).

4. Realm Flutter SDK documentation. URL: https://www.mongodb.com/docs/realm/sdk/flutter/ (дата обращения: 10.04.2026).

5. SQLite documentation. URL: https://www.sqlite.org/docs.html (дата обращения: 10.04.2026).

6. Мартин Р. Чистая архитектура / Р. Мартин. – СПб.: Питер, 2022. – 352 с. – ISBN 978-5-4461-0772-8.