<https://redis.io/commands>

1. **Поясните понятие in-memory Database.**

База данных в памяти ([англ.](https://ru.bmstu.wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) In-memory database, IMDB) — это [система управления базами данных](https://ru.bmstu.wiki/%D0%A1%D0%A3%D0%91%D0%94_(%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%91%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)), которая при хранении компьютерных данных в основном полагается на [оперативную память](https://ru.bmstu.wiki/RAM_(Random_Access_Memory)). Она выделяется на фоне СУБД, в которых используется механизм дискового хранения. Базы данных в памяти быстрее, чем базы данных, оптимизированные для использования дисковых накопителей, поскольку доступ к диску медленнее, чем доступ к памяти, а внутренние алгоритмы оптимизации проще и выполняют меньше инструкций [ЦП](https://ru.bmstu.wiki/CPU_(Central_Processing_Unit)). Доступ к данным из памяти исключает время на поиск при их запросе, что обеспечивает более быструю и предсказуемую работу, чем при использовании дискового накопителя.

Базы данных в памяти предназначены для достижения минимального времени отклика, а также чрезвычайно высокой пропускной способности для систем, критичных к производительности. Это возможно потому, что данные сохраняются и обрабатываются в форме, используемой приложением, что устраняет накладные расходы, связанные с переводом и кэшированием. Технология IMDB способна поддерживать развертывание на уровне приложений, управление данными в режиме реального времени и большинство свойств ACID.

Структуры данных и алгоритмы IMDB предназначены исключительно для обеспечения управления данными, событиями и транзакциями на уровне приложений. По сравнению с полностью кэшированными системами управления реляционными базами данных, IMDB используют значительно меньше ресурсов ЦП. Технология IMDB не использует магнитные диски, как место для хранения первичной базы данных. Вместо этого магнитные диски используются для обеспечения стойкости и восстановления.

Основными областями применения In-Memory решений являются:

1. Анализ данных рынка, Реакция на события (CEP), Торговля.

2. Авторизация, Online транзакции (OLTP).

3. Real-Time аналитика – интерактивное представление данных, витрины данных.

4. Электронная коммерция (Electronic Data Interchange, e-trade, e-cash, e-marketing), персонализация, Real-Time обслуживание.

1. **Поясните понятие хэш-таблица.**

Хеш-табли́ца — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

Хеш-таблицей называется структура данных, предназначенная для реализации ассоциативного массива, такого в котором адресация реализуется посредством хеш-функции. Хеш-функция – это функция, преобразующая ключ key в некоторый индекс i равный h(key), где h(key) – хеш-код (хеш-сумма, хеш) key. Весь процесс получения индексов хеш-таблицы называется хешированием.

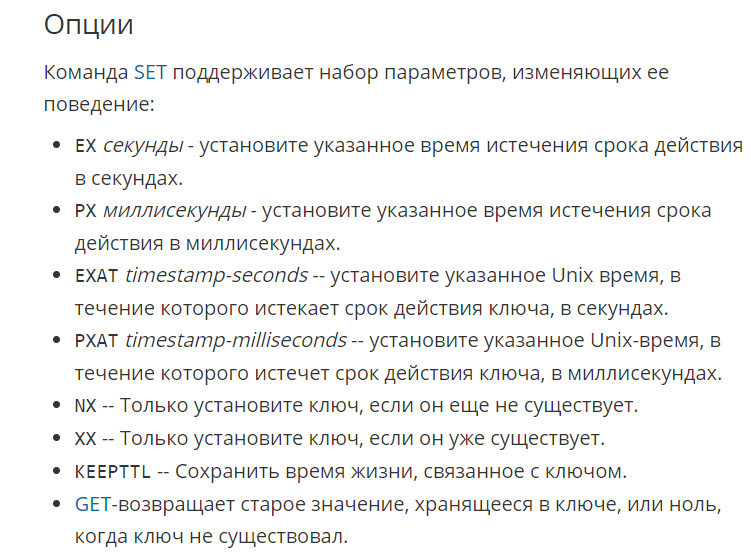
В общем случае хеш-таблица позволяет организовать массив, специфика которого проявляется в связанности индексов по отношению к хеш-функции; индексы могут быть не только целого типа данных (как это было в простых массивах), но и любого другого, для которого вычислимы хеш-коды. Данные, хранящиеся в виде такой структуры, удобны в обработке: хеш-таблица позволяет за минимальное время (O(1)) выполнять операции поиска, вставки и удаления элементов.

1. **Каким образом обеспечивается персистентность данных в СУБД Redis?**

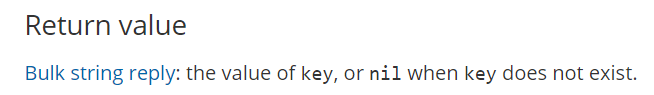
**Персистентность**. Redis сохраняет снимки базы данных на диск. Можно настроить период сохранения данных в зависимости от количества обновленных значений. Также можно использовать режим дозаписи. Для случаев, когда, например, Redis используется в качестве кэша, сохранение на диск можно отключить вовсе.

1. **Поясните назначение команд СУБД Redis: set, get, del, getset.**

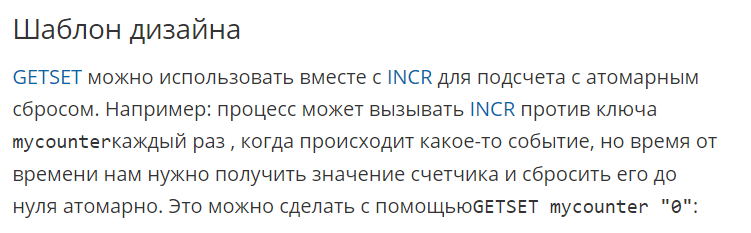
**set** - Установите key, чтобы удерживать строку value. Если key значение уже имеет значение, оно перезаписывается, независимо от его типа. Любое предыдущее время жизни, связанное с ключом, отбрасывается при успешной операции набора.



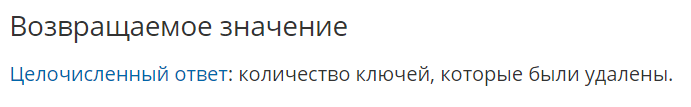
**Get** - Получите значение key. Если ключ не существует, то возвращается специальное значение nil. Ошибка возвращается, если значение, хранящееся в key файле, не является строкой, поскольку [GET](https://redis.io/commands/get) обрабатывает только строковые значения.



**Getset** - Атомарно устанавливает key значение value и возвращает старое значение, хранящееся в key. Возвращает ошибку, если key она существует, но не содержит строкового значения.



**Del** - Удаляет указанные ключи. Ключ игнорируется, если он не существует.

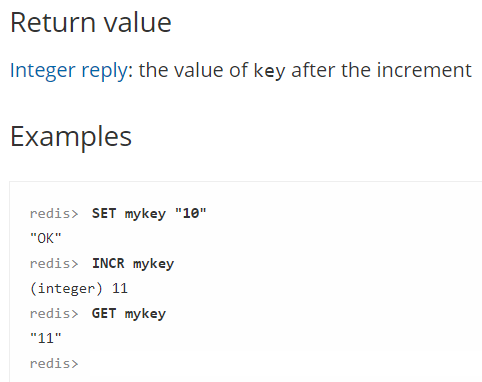


1. **Поясните назначение команд СУБД Redis: incr, decr.**

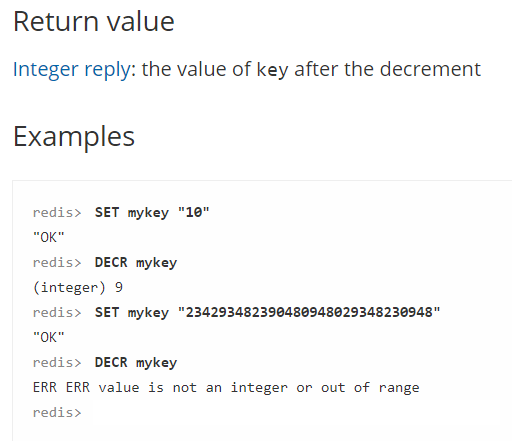
**incr** увеличивает число, хранящееся на key единицу. Если ключ не существует, он устанавливается в 0 значение перед выполнением операции. Ошибка возвращается, если ключ содержит значение неправильного типа или содержит строку, которая не может быть представлена в виде целого числа. Эта операция ограничена 64-битными целыми числами со знаком.

Примечание: это строковая операция, поскольку Redis не имеет выделенного целочисленного типа. Строка, хранящаяся в ключе, интерпретируется как 64-битное целое число со знаком base-10 для выполнения операции.

Redis хранит целые числа в их целочисленном представлении, поэтому для строковых значений, которые фактически содержат целое число, нет никаких накладных расходов на хранение строкового представления целого числа.



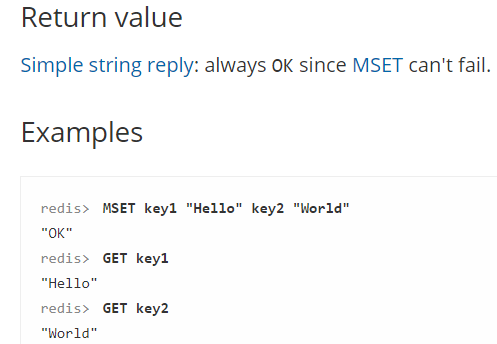
**decr** уменьшает число, хранящееся на key единицу. Если ключ не существует, он устанавливается в 0 значение перед выполнением операции. Ошибка возвращается, если ключ содержит значение неправильного типа или содержит строку, которая не может быть представлена в виде целого числа. Эта операция ограничена **64-битными целыми числами со знаком**.



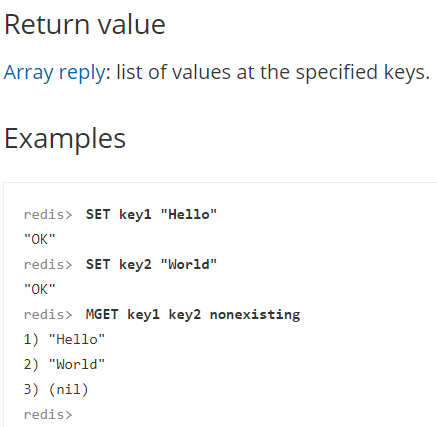
1. **Поясните назначение команд СУБД Redis: mset, mget.**

Устанавливает для данных ключей соответствующие значения. MSET заменяет существующие значения новыми значениями, так же, как и обычный набор. Обратитесь к MSETNX, если вы не хотите перезаписывать существующие значения.

MSET является атомарным, поэтому все заданные ключи устанавливаются сразу. Клиенты не могут видеть, что некоторые ключи были обновлены, в то время как другие остаются неизменными.



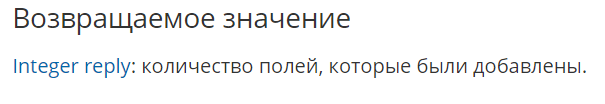
**mget** - Возвращает значения всех заданных ключей. Для каждого ключа, который не содержит строкового значения или не существует, возвращается специальное значение nil. Из-за этого операция никогда не заканчивается неудачей.



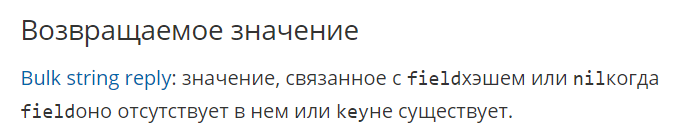
1. **Поясните назначение команд СУБД Redis: hset, hget.**

Наборы field в хэше, хранящемся в key to value. Если key он не существует, создается новый ключ, содержащий хэш. Если field хэш уже существует, он перезаписывается.

Начиная с Redis 4.0.0, HSET является вариативным и позволяет использовать несколько field/value пар.



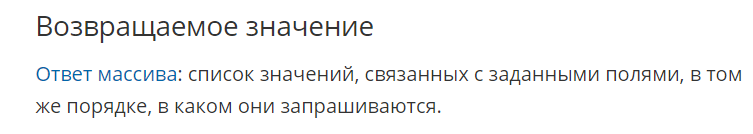
hget - Возвращает значение, связанное с field в хэше, хранящемся на key.

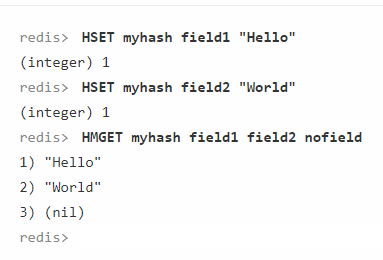


1. **Поясните назначение команд СУБД Redis: hmset, hmget.**

Возвращает значения, связанные с указанным fields в хэше, хранящемся на key.

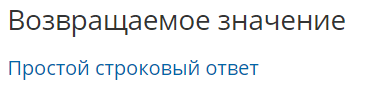
Для каждого field, что не существует в хэше, nil возвращается значение. Поскольку несуществующие ключи рассматриваются как пустые хэши, запуск HMGET против несуществующего key вернет список nil значений.

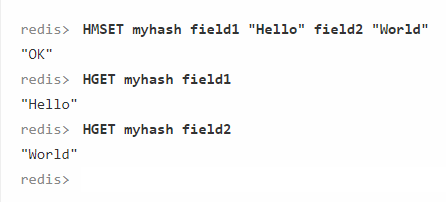




Устанавливает для указанных полей соответствующие значения в хэше, хранящемся на key. Эта команда перезаписывает все указанные поля, уже существующие в хэше. Если key он не существует, создается новый ключ, содержащий хэш.

Согласно Redis 4.0.0, HMSET считается устаревшим. Пожалуйста, предпочтите HSET в новом коде.





1. **Поясните назначение команды СУБД Redis: exists.**

Возвращает, если key существует.

Начиная с Redis 3.0.3, можно указать несколько ключей вместо одного. В этом случае он возвращает общее количество существующих ключей. Обратите внимание, что возврат 1 или 0 для одного ключа-это всего лишь частный случай вариативного использования, поэтому команда полностью обратно совместима.

Пользователь должен знать, что, если один и тот же существующий ключ упоминается в аргументах несколько раз, он будет подсчитан несколько раз. Так что если somekey существует, EXISTS somekey somekey, то вернет 2.

