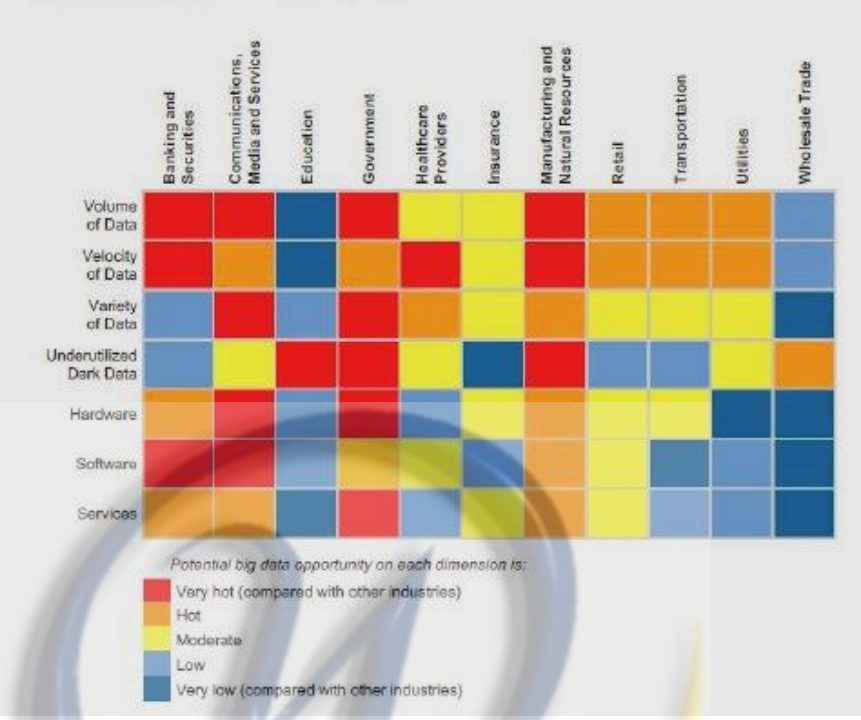
# BAB II

# LANDASAN TEORI

## 2.1 Memahami Teknologi NoSQL

Menurut *IBM*, Beberapa *non relational database technology* sudah muncul, dan *NoSQL* adalah nama penggambaran untuk semuanya. Yang secara terus-menerus meningkat di aplikasi *web*, *mobile*, dan *IoT* (Internet of Things) di sisi tren kemunculan manfaat konsumen *online* dan kelas data yang baru, menyebabkan pengembang untuk me-evaluasi bagaimana datanya disimpan dan ditangani. Pengembang sepertinya menemukan kesulitan itu, dan kadang-kadang bahkan mustahil, cepatnya menggabungkan semua data ini ke dalam model relasi saat mengatur *level* performa pengguna langsung. Ini menyebabkan banyak dilihat saat *database NoSQL* untuk fleksibilitas yang ditawarkan, dan alasan terbesar kenapa pasar global *NoSQL* meramalkan akan meraih *USD* $ 3.4 juta di tahun 2020 [26].

Menurut dari dokumentasi *Mongodb*, teknologi yang dipakai di masa lalu (Mysql) tidak pernah didesain untuk kecepatan dan skala permintaan oleh aplikasi modern, *NoSQL* itu hanya mendaur-ulang teknologi yang tidak pernah dipakai oleh *MySQL*. Banyak di perusahaan teknologi yang berperan sebagai arsitek ataupun pengembang lebih mengandalkan pendekatan baru hanya untuk manajemen *database* dan analisis namun lebih menghindari dari resiko kompleksitas di masa yang akan datang dari segudang teknologi yang masing-masing membutuhkan alat sendiri dan cara kerjanya [2]. Dengan mengerti kenapa secara terbuka mengadopsi teknologi *NoSQL* dan apa masalah perusahaan yang telah bergulat untuk mengimplementasikan dengan caranya sekarang. Dari pandangan perusahaan industri global, *Gartner Group* memproduksi kisi yang disuplai ke pemandangan yang menarik bagaimana berbagai macam industri bisa melayani teknologi *NoSQL* diberikan karakteristik kunci itu dalam definisi *big data* [8].




**Gambar 2.1** Industri *heat map* untuk *big data* oleh *Gartner Group* [8].

Dari beberapa aspek tersebut, teknologi *database* memiliki lebih dari permintaan seperti [16];

a. Membaca tingkat tinggi secara bersamaan dan menulis dengan latensi rendah

   *Database* yang permintaan untuk menulis kebutuhan yang tinggi bersamaan membaca dan menulis dengan latensi rendah, saat yang sama, dalam rangka untuk lebih meningkatkan kepuasan pelanggan, permintaan basis data untuk membantu aplikasi bereaksi cukup cepat.

b. *Efficient big data storage* dan *access requirements*

   Besaran aplikasi, seperti *SNS* dan mesin pencari, perlu *database* untuk memenuhi penyimpanan data yang efisien dan dapat merespon kebutuhan jutaan lalu lintas.

c. *High scalability* and *high availability*

Dengan meningkatnya jumlah permintaan bersamaan dan data, *database* perlu untuk dapat mendukung mudah ekspansi dan *upgrade*, dan memastikan cepat layanan tanpa gangguan.

d. *Lower management* and *operational costs*

Dengan peningkatan dramatis dalam data, biaya *database*, termasuk biaya *hardware*, biaya perangkat lunak dan biaya operasi, telah meningkat. Oleh karena itu, perlu biaya yang lebih rendah untuk menyimpan data yang besar.

Meskipun *database* relasional telah menduduki tinggi posisi di area penyimpanan data, tetapi ketika menghadapi atas persyaratan, memiliki beberapa keterbatasan.

e. *Slow reading and writing*

Sebuah *relational database* sendiri memiliki logika kompleksitas tertentu, dengan peningkatan ukuran data, dan rawan membawa *deadlock* dan masalah konkurensi lainnya, ini telah menyebabkan penurunan cepat dalam efisiensi membaca dan menulis.

*f. Limited capacity*

Ada *relational database* tidak dapat mendukung data besar di mesin pencari, *SNS* atau *Big system*.

*g. Expansion difficult*

Mekanisme korelasi multi-tabel yang ada di *relational database*, menjadi faktor utama skalabilitas *database*.

## 2.2 Jenis Data Model

Cara utama dimana *database non-relational* berbeda dari *database relational* adalah model data. Meskipun ada puluhan *database non-relational*, mereka terutama jatuh menjadi salah satu dari tiga kategori berikut [9];

a. *Document Model*

Sedangkan *relational database* menyimpan data dalam baris dan kolom, *document database* menyimpan data dalam *document*. *Document* ini biasanya menggunakan struktur yang seperti *JSON* (Javascript Object Notation), *format* populer diantara para pengembang. Dokumen memberikan intuitif dan cara alami untuk model data yang berkaitan erat dengan pemrograman berorientasi obyek dengan setiap *document* adalah objek efektif. *Document* berisi satu atau lebih bidang, dimana masing-masing bidang berisi nilai diketik, seperti *string*, *date*, *binary*, ataupun *array*. Daripada menyebar keluar *record* di beberapa kolom dan tabel terhubung dengan *foreign key*, setiap *record* dan yang terkait (misalnya *related*) data biasanya disimpan bersama-sama dalam satu *document*. Ini menyederhanakan akses data dan dalam banyak kasus, eliminasi perlu lagi penggunaan operasi *JOIN* dan kompleks transaksi multi *record*.

Dalam *document database*, gagasan skema adalah dinamis, setiap dokumen dapat berisi bidang yang berbeda. Fleksibilitas ini dapat sangat membantu untuk pemodelan data tidak terstruktur dan data *polymorphic* (data *polymorphic* artinya dapat menimpa (override) suatu *method*, yang berasal dari *parent class* (super class) dimana *object* tersebut diturunkan, sehingga memiliki kelakuan yang berbeda). Hal ini juga membuat lebih mudah melibatkan aplikasi selama pengembangannya seperti

menambahkan bidang baru. Selain itu, *document database* umumnya memberikan ketangguhan *query* yang pengembang telah harapkan dari *relational database*. Khususnya, data dapat dilihat berdasarkan kombinasi bidang dalam dokumen.

Aplikasi: *document database* yang tujuan umum berguna untuk berbagai macam aplikasi karena fleksibilitas dari model data, kemampuan untuk *query* pada setiap bidang dan pemetaan alami dari model data dokumen ke objek di bahasa pemrograman modern.

Contoh *MongoDB* dan *CouchDb*.

b. *Graph Model*

*Graph database* menggunakan struktur grafik dengan *node, edge* dan *properties* untuk mewakili data. Pada dasarnya, data dimodelkan sebagai jaringan hubungan antara tertentu elemen. Sementara model grafik mungkin menjadi kontra-intuitif dan mengambil beberapa saatnya untuk mengerti, dapat berguna untuk kelas khusus *query*. Daya tarik utama adalah bahwa hal itu membuatnya mudah untuk model dan menavigasi hubungan antara entitas dalam sebuah aplikasi.

Aplikasi: *Database graph* berguna dalam kasus di mana hubungan yang melintasi adalah inti untuk aplikasi, seperti navigasi sosial koneksi jaringan, topologi jaringan dan rantai pasokan.

Contoh *Neo4j* dan *Giraph*

c. *Key-Value* dan *Wide Column Model*

Dari perspektif model data, *key-value store* adalah jenis yang paling dasar dari basis data *non*-relasional. Setiap *item* dalam *database* disimpan sebagai nama atribut,

atau *key*, bersama-sama dengan nilainya. Nilai, bagaimanapun, adalah sepenuhnya buram ke sistem; data hanya dapat ditanyakan oleh *key*-nya. Model ini dapat berguna untuk mewakili *polymorphic* dan 2 data tidak terstruktur, sebagai *database* tidak memberlakukan *set* skema di seberang sepasang *key-value*.

*Wide column store*, atau *column family store*, memakai yang tersebar dimana-mana, didistribusikan peta multi-dimensi diurutkan untuk menyimpan data. Setiap *record* dapat bervariasi dalam jumlah kolom yang disimpan. Kolom dapat dikelompokkan bersama-sama untuk akses di *column family*, atau kolom dapat tersebar di beberapa *column family*. Data diambil oleh *primary key* per *column family*.

Aplikasi: *key value store* dan *wide column store* berguna untuk satu *set* aplikasi yang sempit hanya data *query* dengan satu *key-value*. Daya tarik dari sistem ini adalah kinerja dan skalabilitas mereka, yang dapat sangat dioptimalkan karena keserhanaan pola akses data dan *opacity* (sifat tidak jelas) dari data itu sendiri.

Contohnya: *Riak* dan *Redis* (key-value), *Hbase* dan *Cassandra* (wide column).

Berikut dengan kumpulan beberapa jenis *NoSQL* dengan kemampuannya masing-masing:

**Tabel 2.1** Kumpulan *Database NoSQL* serta tipe kemampuannya [1].

| Tipe *NoSQL* | Nama *Database NoSQL* |
|---|---|
| *Unresolved* dan *uncategorized* | *Kribybase, Tokutek, Recutils, FileDB, CodernityDB* |
| *Scientific* dan *specialized* | *BayesDB, GpuDB* |
| Relasi *NoSQL* yang lain | *IBM lotus, ExtremeDB, RDM embedded, ISIS family, Moonshadow, VaultDB, Yserial.* |
| *Streaming series* | *Axibase, Influxdata, PipelineDB, KDB+* |
| *Event sourcing* | *Event store, Eventsourcing for Java* |
| *Multivalue database* | *U2, Openinsight, Tigerlogic pick, Reality, Openqm* |
| *Multidimensional* | *Globals, Intersystems cache, GT.m, SciDB, Minim DB, Rasdaman, DaggerDB* |
| *XML* | *Emc documentum XDB, Exist, Sedna, Basex, Qizx, Berkeley DB XML* |
| *Grid & Cloud Database* | *Crate data, Oracle Coherence, Gigaspaces, Gemfire, Infinispan,* |

| Tipe *NoSQL* | Nama *Database NoSQL* |
|---|---|
| | *Queplix, Hazelcast* |
| *Object Database* | *Versant, DB4O, Objectivity, Gemstone/s, Starcounter, Perst, VelocityDB, HSS database, ZoDB, Magma, Neo, SaqoDB, Jade, Sterling, MarcelloDB, Morantex, Eyedb, Framerd, Ninja Database Pro, Ndatabase, Picolisp, Acid-state, ObjectDB, Coreobject* |
| *Multimodel* | *ArangoDB, OrientDB, Datomic, GunDB, CortexDB, AlchemyDB, WonderDB, RockallDB, FoundationDB* |
| *Graph* | *Neo4j, RangoDB, OrientDB, Infinite Graph, Sparksee, Titan, Infogrid, Hypergrid, HypergraphDB, Graphbase, Trinity, Allegrograph, BrightstarDB, Big Data, Meronymy, WhiteDB, Onyx database, Openlink Virtuoso, VertexDB, FlockDB, Weaver, BrightstarDB, Execom iog, Fallen 8, MarkLogic, Stardog* |
| *Key value* | *DynamoDB Amazon, Redis, LevelDB,* |

| Tipe *NoSQL* | Nama *Database NoSQL* |
|---|---|
| | *BerkeleyDB, Aerospike, Couchbase, Faircom c-treeACE, HyperDex, MemcacheDB, MUMPS, Oracle NoSQL DB, Riak, BigTable Google. Voldemort LinkedIn* |
| *Document Store* | *MongoDB, Elasticsearch, Couchbase Server, CouchDB, RethinkDB, Clusterpoint, DocumentDB, IBM Domino* |
| *Wide column store* | *Hadoop, Cassandra, Mapr, Hortonworks, Accumulo, Druid, Hbase, Vertica* |

## 2.3 Kelebihan Dari NoSQL

Kelebihan utama dari *NoSQL* seperti;

a. Menyediakan berbagai macam model data yang dapat dipilih.

b. Skalabilitas yang mudah.

c. *Database Administrator* tidak dibutuhkan.

d. Beberapa *NoSQL Database as a Service* (DbaaS) membuktikan seperti *Riak* dan *Cassandra* diprogram untuk menangani kesalahan *hardware*.

e. Tercepat, lebih efisien dan fleksibel.

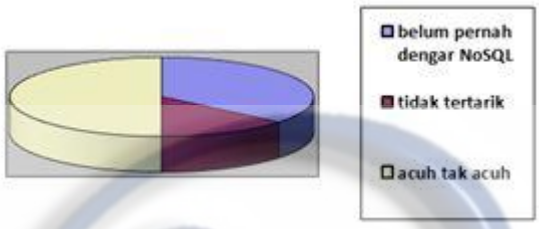
f. Telah terlibat saat kecepatan yang sangat tinggi [37].

## 2.4 Kekurangan NoSQL

Dibandingkan dengan *relational database* pada umumnya, meskipun *database NoSQL* banyak sekali keuntungan, ada beberapa kekurangan juga [15].

a. *Complexity*; karena *database NoSQL* tidak bisa dengan *SQL*, diperlukan pemograman *manual query*, yang akan cepat untuk penugasan yang sederhana namun memakan waktu yang lain. Selain itu, pemograman *query* yang kompleks untuk *database* mungkin sulit untuk diterapkan.
b. *Reliability*; relasi *database native* mendukung *ACID* (A: *automicity*, C: konsistensi, I:isolasi, D:daya tahan). Sementara *database NoSQL* tidak. Dengan demikian, *database NoSQL* tidak *native* menggunakan tingkat kehandalan yang *ACID* sediakan. Jika *user* ingin *database NoSQL* menerapkan pembatasan *ACID* ke *dataset*, perlu melakukan pemograman tambahan.
c. Konsistensi; sejak *database NoSQL* tidak bisa *native* mendukung transaksi *ACID* tanpa program tambahan, mereka memiliki kekurangan dalam konsistensi. Tidak menyediakan konsistensi memungkinkan kinerja yang lebih baik dan skalabilitas tetapi masih menjadi masalah untuk beberapa jenis aplikasi dan transaksi, seperti yang terlibat dalam perbankan.
d. Ketidaktahuan dengan teknologi; sebagian besar organisasi tidak terbiasa dengan *database NoSQL*, sehingga mereka mungkin tidak merasa itu cukup luas untuk memilih satu atau bahkan untuk menentukan bahwa apakah pendekatan mungkin

yang sesuai untuk tujuan mereka. Dari gambar 2.2, jelas bahwa 44% dari pengguna bisnis belum pernah mendengar tentang *NoSQL*, 17 persen pengguna bisnis tidak tertarik *NoSQL*. Yang mengatakan 61 % dari pengguna bisnis acuh tak acuh terhadap *NoSQL*.




**Gambar 2.2** *Diagram* ketidaktahuan akan teknologi [15].

## 2.5 Perbandingan antara NoSQL dan MySQL

Pengolahan *database* dilakukan menggunakan bahasa *MySQL*. Bahasa ini menjadi salah satu pondasi dari kelahiran dari *database*. Tetapi *trend* saat ini arah penggunaan *MySQL* mulai tergerus dengan kehadiran *NoSQL*. Menurut komunitas *MongoDB*, Penggunaan *NoSQL* mulai menggerus keberadaan *MySQL*, *Oracle*, *SQL Server*. Hal ini dikarenakan penggunaan yang lebih mudah dan kemampuan yang lebih baik, sebagai contoh penggunaan *Hadoop* sebagai basis data dalam skala besar mulai menggantikan *Oracle* dan *SQL Server*, dan *MongoDB* mulai bersaing ketat dengan penggunaan *MySQL* [1].

*NoSQL* dapat secara jelas *No Es Queue El* atau *Not only SQL*. Arti dari *NoSQL* adalah teknologi yang menandingi *MySQL*. Maksudnya dalam penggunaan *database* tidak hanya menggunakan struktur *MySQL* yang sudah ada, akan tetapi bisa menggunakan cara pengambilan data secara langsung tanpa mengikuti aturan yang sudah ada tetapi dengan cara *indexing* di masing-masing *table* (*field* yang diambil) [1][3].

Sedangkan menurut dari Yishan Li, Penggunaan *NoSQL* saat ini tidak ditujukan untuk menjadi pengganti dari Basis data *relational* atau lebih dikenal dengan *SQL*. Karena *NoSQL* dikembangkan untuk memecahkan masalah yang dihadapi oleh *database* dengan skema relasional. Untuk membedakannya tersusun dalam bentuk tabel [1][3];

**Tabel 2.2** Perbandingan *NoSQL* dan *SQL* [1][3].

| | *Database SQL* | *Database NoSQL* |
|---|---|---|
| Jenisnya | Satu tipe *SQL database* dengan variasi *minor* | Banyak jenis perbedaan termasuk *key-value store*, *document database*, *wide-column store*, dan *graph database*. |
| Sejarah pengembangan | Dikembangkan tahun 1970 mengurus gelombang pertama aplikasi data *storage* | Dikembangkan di akhir 2000 mengurus batasan *database SQL*, khususnya *scalability*, *multi-structured data*, *geo distribution* dan *agile development sprints*. |
| Contohnya | *MySQL*, *Postgres*, *Microsoft SQL Server*, *Oracle database* | *MongoDB*, *Cassandra*, *Hbase*, *Neo4j*, *Amazon Dynamodb*. |
| Model data *storage* | Individual *record* (contohnya 'karyawan') di stored sebagai *row* di tabel, dengan setiap kolom di *storing* ke spesifik potongan data mengenai *record*nya (contohnya 'manajer','tanggal dipekerjakan' dll), lebih seperti *spreadsheet*. Hubungan data di *stored* dipisah tabel, dan kemudian bergabung bersama ketika lebih kompleks *query* nya dieksekusi. Sebagai contoh 'kantor' kemungkinan di *store* di 1 tabel, dan 'karyawan' di yg lain. Ketika *user* ingin menemukan alamat kantor kerja pada karyawan, *engine* | Variasi berdasarkan tipe data. Contohnya fungsi *key-value stores* mirip ke *database SQL*, tapi hanya memiliki 2 kolom ('*key*' dan '*value*'), dengan lebih kompleks informasi kadang-kadang di *store* sebagai BLOB dengan kolom '*value*'. Dokumen *database* melakukan diluar dengan tabel dan *row* model bersama, men-*storing* semua data yang bersangkutan bersama dalam *single* '*document*' di JSON, XML, atau *format* lain, yang mana sekumpulan bisa bernilai hirarki. |

| | *Database SQL* | *Database NoSQL* |
|---|---|---|
| | *database* gabung tabel 'karyawan' dan 'kantor' bersama untuk mendapatkan semua informasi yang diperlukan. | |
| *Schema* | Stuktur dan jenis data diperbaiki dalam kemajuannya. Untuk *store* informasinya mengenai data item terbaru, semua *database* harus di ubah, selama yang mana waktu *database* harus di *offline* kan. | Tipikal dinamis, dengan beberapa menegakkan aturan data validasi. Aplikasi bisa menambah *field* baru dan tidak seperti *SQL row* tabel, tidak sama data bisa di *store* diperlukan bersama. Untuk beberapa *database* (contoh *wide column store*) entah mengapa lebih menantang untuk menambah *field* baru secara dinamis. |
| *Scalling* | Secara *vertical*, maksudnya satu *server* mungkin dibuat meningkatnya lebih bertenaga agar berhadapan dengan meningkatnya permintaan. Ini mustahil untuk menyebarkan *database SQL* lebih banyak *server*, tapi signifikan menambahkan teknisnya yang umumnya dibutuhkan, dan inti fitur relasi seperti *JOIN*, integrasi referensi dan transaksi bertipikal hilang. | Secara *horizontal*, maksudnya menambah kapasitas, *database administrator* bisa dengan sederhana menambahkan komoditas *server* ataupun *cloud* contohnya. *Database* secara otomatis menyebarkan data lintang ke *server* bila diperlukan. |
| Model pengembangan | Bercampur dalam *open source* (contohnya *postgres, MySQL*) *open* | *Open source* kecuali yang dimiliki perusahaan *amazon* (*DynamoDB*) itu |

| | *Database SQL* | *Database NoSQL* |
|---|---|---|
| | *source* dan *close source* (contohnya *Oracle*) | berbayar. |
| *Support transaction* | Ya, *update* bisa dikonfigurasikan ke semuanya dengan sempurna ataupun tidak | Keadaan tertentu dan *level* tertentu (contoh *level document* vs *level database*) |
| Manipulasi data | Bahasa yang spesifik menggunakan *statement select*, *insert*, dan *update* contohnya *select fields from table where* | Melalui *Object Oriented API* |
| Konsistensi | Bisa dikonfigurasikan untuk konsistensi yang kuat | Tergantung pada produk. Beberapa menyediakan konsistensi yang kuat (contoh *mongodb*, dengan menyesuaikan konsistensi untuk membaca) padahal yang lain menawarkan bahkan lebih konsisten (contoh *cassandra*). |

## 2.6 Gambaran Ikhtisar NoSQL

*Database NoSQL* bisa memproses berbagai data apapun hingga dapat menyimpan data lebih dari kemampuan daya penyimpanan dari *MySQL*. Salah satu perusahaan yang sudah menerapkan *NoSQL* adalah *twitter* untuk menganalisis miliaran *tweet* data dan *trending tweet* langsung, dan juga perusahaan *Alphabet* dengan produk *Google* dan *Youtube* yang menampung triliunan video juga *link* mesin pencari, ada lagi *Facebook* yang menurut Harrison Fisk yang bekerja di *Facebook* bagian dari tim *database performance* bahwa *facebook* memakai *database NoSQL* yang bervariasi seperti *HBASE* dipakai untuk pesan dan monitoring lalu *Hadoop/Hive* sebagai *primary big data analytical store* dan juga *memcached* masih selalu dipakai.

Menurut Maria Teresa Gonzales dan temannya berpendapat *database NoSQL* dipakai untuk berbeda aplikasi dan memiliki perbedaan tipe termasuk *document store*, *column families*, *key value*, *graphs* dan *multimodel database*. Di dalam riset ini difokuskan untuk *key value database*. Tentunya banyak layanan internet yang memakai didesain dan diimplementasikan menggunakan *key value database* seperti daftar penjual terbaik, ataupun keranjang belanja lalu katalog produk yang biasa ada di dalam *website amazon.com*. di sini ada beberapa fitur dari beberapa *key value database* seperti *redis, voldemort, dynamo* dan *riak* [21].

**Tabel 2.3** Menampilkan fitur *NoSQL key value database* [21].

| *Features* | *Redis* | *Voldemort* | *Dynamo* | *Riak* |
|---|---|---|---|---|
| *Transactions* | *Execution of group of commands* | *No* | *No* | *No* |
| *Roll back operation* | *No* | *No* | *No* | *No* |
| Penyesuaian konsistensi | *Yes* | *Yes* | *Yes* | *yes* |

## 2.7 Jenis – Jenis NoSQL

Ada beberapa jenis *NoSQL* menurut Ameya Nayak dan rekannya, berikut penjelasannya seperti [37];

*a. Big Table*

*Google big table* adalah yang ditekan *database* pada performa tinggi yang mana awal dirilis tahun 2005 dan dibangun pada *Google File System* (GFS). Dikembangkan menggunakan C dan C++. Menawarkan konsistensi, toleransi kesalahan dan keteguhan. Di desain untuk skala sepanjang ratusan mesin dan mudah menambah lebih mesin juga. *Big Table* di implementasikan memiliki tiga pokok komponen: *library* yang terhubung ke dalam setiap *client*, satu *master server*, dan banyak *tablet server*. *Tablet server* dipakai mengelola *set tablet* (sama seperti tabel dalam *RDBMS*). *Master server* menangani perubahan *schema*, tugas performa seperti menetapkan *tablet* ke *tablet server*, menyeimbangkan beban *tablet server*, mengeluarkan sampah data dan lain sebagainya. *Big Table* tidak terdistribusi di luar *Google*, tapi tersedia sebagai bagian *Google app engine*. *Big Table* dipakai oleh beberapa jumlah aplikasi *Google* seperti *Gmail*, *YouTube* dan *Google Earth*.

*b. Cassandra*

*Cassandra* dikembangkan oleh *Apache Software Foundation* dan di rilis tahun 2008. Dikembangkan menggunakan *Java*. Berdasarkan pada kedua *Amazon*

*DynamoDB* model dan *Google Big Table*. Dengan demikian terlibat konsep keduanya penyimpanan *key-value* dan penyimpanan *column*. Menawarkan fitur seperti ketersediaan tinggi, toleransi patisi, persistensi, skalabilitas tinggi dan lainnya. Ada *Schema* yang dinamis. Bisa dipakai untuk aplikasi yang bervariasi seperti *website* jaringan sosial, bank dan keuangan, data analisis yang *real time*, *online retail* dan lainnya. *Cassandra* sudah dipakai oleh *Adobe, Digg, eBay, Twitter* dan lainnya. Kekurangan *Cassandra* adalah *read* secara komparatif masih lambat dibandingkan *write* nya.

c. *MongoDB*

*MongoDB* dikembangkan *10gen* dan awal di rilis tahun 2009. dikembangkan menggunakan 2009. Dikembangkan menggunakan C++. Ber-performa tinggi dan efisien *database*. Dibuktikan fitur seperti konsistensi toleransi kesalahan, persistensi. *MongoDB* membuktikan penambahan fitur seperti agregasi, *ad hoc query*, *indexing, auto sharding* dan lainnya. Di dalam *MongoDB, document* utamanya disimpan di *BSON* (Binary JSON) *format*. *BSON document* berisi daftar pesanan elemen yang konsisten pada bidang nama, jenis dan nilai. *BSON* efisien pada dalam kedua ruang penyimpanan dan kecepatan *scan* ketika dibandingkan ke *JSON*. *MongoDB* dipakai *GridFS* sebagai spesifikasi untuk *file* penyimpanan besar. *MongoDB* juga cocok untuk aplikasi seperti konten manajemen sistem, arsip, *real time analytics* dan lainnya. *MongoDB* saat ini dipakai oleh *MTV networks*, *Foursquare*, *The Guardian News* dan lainnya. Juga

dipakai dalam proyek seperti *CERN LHC*, *UIDAI Aadhaar* yang mana India *unique identification project*. Kekurangannya adalah tidak bisa diandalkan dan *indexing* banyak mengambil memori *ram*.

*d. CouchDB*

*CouchDB* dikembangkan oleh *Apache Software Foundation* dan awalnya di rilis tahun 2005. Dikembangkan memakai C++. Memakai *JSON document* untuk menyimpan data dan dibuktikan *RESTful HTTP API* untuk membuat dan meng-*update database document*. Dibuktikan *Javascript* sebagai bahasa *query*. Dibuktikan membangun *web* aplikasi dinamakan *FULTON* yang mana bisa dipakai untuk administrasi. Ketersediaan tinggi, toleransi kesalahan dan persistensi. Diimplementasikan *Multi-Version Concurrency Control* (MVCC) dengan demikian membuktikan yang bersamaan akses ke *user*. *CouchDB* memiliki replikasi yang sangat bagus dan kemampuan tersinkronisasi. Bisa dipakai untuk aplikasi yang terlibat kadang mengubah data didefinisikan *query* yang perlu dipakai. Bisa juga dipakai dalam kasus dimana koneksi jaringan mungkin boleh atau mungkin tidak tersedia, tapi aplikasi harus tetap jalan, seperti dalam kasus perangkat *handphone* berdasarkan aplikasi. Bisa juga dipakai untuk *CRM* (Customer Relationship Management) dan sistem (Content Management System) *CMS*. *CouchDB* dipakai oleh banyak *website* seperti LotsOfWords.com dan Friendpaste.com juga oleh *facebook app* seperti *Horoscope*, *Brithday Greeting Cards* dan lainnya. Beberapa *Couchdb* sementara dilihat dalam besar

*dataset* benar-benar lambat, tidak bagus berurusan dengan relasi data, tidak ada dukungan untuk *ad-hoc query*.

*e. Neo4j*

*Neo4j* dikembangkan oleh *Neo Technology* dan awal rilis tahun 2007. Dikembangkan menggunakan *Java*. Tingkat performa tinggi *database* grafik yang mana dibuktikan *object oriented,* fleksibel struktur jaringan. Berdasarkan *property* graph data model yang mana terdiri dari *node* dan hubungan sepanjang dengan *properties* nya. *Reliable*, mengalah pada *ACID*, ketersediaan tinggi dan skalabilitas. Menawarkan *REST interface* dan *Java API* tenang nyaman dipakai. Bisa juga ditempelkan ke dalam *jar files*. Memakai *CHYPER* sebagai bahasa *query*. *Neo4j* harus dipakai dalam perangkat lunak yang terlibat hubungan kompleks seperti jaringan sosial, rekomendasi mesin dan lainnya. *Sharding* tidak mungkin dalam *Neo4j* harus dikurangi jika relasinya tidak ada bersama data. Beberapa perusahaan bahwa yang memakai *Neo4j* adalah *Adobe, Accenture, Cisco, Luthfansa, Telenor* dan *Mozilla*.

*f. DB4o*

*DB4o* dimulai oleh Carl Rosenberger tahun 2000 dan produk di luncurkan tahun 2001. Pada tahun 2004 dikomersialkan oleh *DB4Objects Inc* dan di akuisisi oleh *Versant Corporation* pada tahun 2008. *db4o* dikembangkan menggunakan *Java* dan *C#*. Dibuktikan *GUI* dinamakan *Object Manager Enterprise* (OME) yang

mana bisa dipakai untuk berbagai tujuan seperti koneksi *database*, *browsing database*, membangun *query* dan bahkan fungsi administrasi. Dibuktikan *Native Query* (NQ) yang mana membolehkan *user* memakai bahasa *object oriented programming* yang umum seperti *Java*, *C#*, atau *VB.net* daripada bahasa *query* seperti *SQL*. Dibuktikan fungsi yang membolehkan *user* menyimpan *object* dalam satu perintah juga dibuktikan *db4o*. Sistem replikasi yang membolehkan sinkronisasi relasi dengan *db4o*. Utama dari penarikan atau kekurangan *db4o* adalah tidak dibuktikan membangun dalam mendukung ekspor atau impor data dari *JSON, XML* atau *text file* yang mana dibuktikan penyimpanan data lain. Ini tidak dibuktikan fitur seperti referensi integritas, alat *OLAP* ditawarkan oleh *SQL*. beberapa perusahaan yang memakai *db4o* adalah *BMW, Bosch, IBM, Intel*, dan *Seagate*.

## 2.8 Gambaran Ikhtisar MySQL

*Mysql* adalah kumpulan data yang terstruktur. Bisa melakukan apapun dari menyimpan daftar penyimpanan seperti galeri foto atau bermacam-macam informasi dari pelbagai jaringan. Untuk melakukan menambah, mengakses dan memproses penyimpanan data di komputer dengan mengatur jumlah data yang besar, gunakanlah *database management system* seperti *server MySQL*.

Penyimpanan *MySQL* pada setiap *database* dinamakan *schema*. Saat membuat tabel, tabel penyimpanan *MySQL* dalam bentuk *file .frm* nama yang sama seperti tabel. Dengan demikian, ketika membuat tabel dinamakan *MyTable*, *MySQL* menyimpan

tabel dinamakan *MyTable.frm*. Karena *MySQL* memakai sistem *file* untuk menyimpan nama *database* dan tabel, tergantung *platform*-nya. Ada beberapa *engine* penyimpanan yang dimiliki oleh *MySQL* menurut Tkachenko dan rekannya seperti [38]:

*a. InnoDB Engine*

Penyimpanan *InnoDB* seri data satu atau lebih data *file* yang secara kolektif diketahui sebagai *tablespace*. *Tablespace* diketahui secara esensial *black box* yang *InnoDB* kelola semuanya sendiri. *InnoDB* bisa menyimpan setiap data tabel dan *indexes* di dalam *file* terpisah. *InnoDB* bisa juga memakai partisi *disk* mentah untuk membanung *tablespace*, tapi *filesystem* yang modern tidak diperlukan. Tabel *InnoDB* dibangun pada *clustered index*, terbukti sangat cepat *primary key* walaupun *secondary indexes* (*index* yang bukan *primary key*) berisi kolom *primary key*, jika *primary key* besar, *index* yang lain akan menjadi besar.

*b. MyISAM Engine*

*MyISAM* jenis penyimpanan setiap tabel dalam dua *file*. data *file* dan *index file*. dua *file* seperti ekstensi *.MYD* dan *.MYI*. Tabel *MyISAM* bisa berisi dinamis atau statis. Jumlah kolom pada tabel *MyISAM* bisa tertahan secara terbatas oleh ketersediaan ruang *disk* pada *server database user* dan besaran *file* yang dibuat untuk dioperasikan sistem.

c. *Archive Engine*

*Archive engine* hanya mendukung *query INSERT* dan *SELECT*, dan tidak mendukung *index* sampai *MySQL* yang terbaru. Menyebabkan *disk*nya berkurang daripada *MyISAM*, karena penyangga data *write* dan dikompres setiap baris dengan *zlib* seperti yang dimasukkan. Juga, setiap *query SELECT* membutuhkan pengamatan semua tabel. Dengan demikian *archive table* terbaik untuk perolehan data, dimana analisis pengamatan ke semua tabel, atau ingin cepat pada *query INSERT*. *Archive* mendukung *row-level locking* dan penyangga sistem yang khusus untuk konkurensi yang tinggi. Mesin penyimpanan yang optimalisasi untuk *insert* yang kecepatan tinggi dan kompresi penyimpanan.

d. *Blackhole Engine*

*Blackhole engine* tidak memiliki semua mekanisme penyimpanan. Membuang setiap *INSERT* sebagai ganti penyimpanannya. Walaupun, *query server write* melawan tabel *blackhole* untuk *log*, jadi bisa direplikasi atau tetap dalam *log*. Itu membuat mesin *blackhole* populer untuk pemasangan pagar replikasi dan audit *log*, meskipun terlihat masalah cukup disebabkan oleh setiap pemasangan yang tidak direkomendasikan.

e. *CSV Engine*

*CSV engine* bisa diterjemahkan sebagai *comma-separated values* (CSV) *file* sebagai tabel, tapi bukan mendukung *index*. Mesin ini memperbolehkan *copy file*

ke dalam dan ke luar *database* saat *server* berjalan. Jika di ekspor *file CSV* dari *spreadsheet* dan menyimpan ke dalam *server* data tujuan *MySQL*, *server* bisa membaca langsung. Persamaannya, jika *write* data ke tabel *CSV*, program eksternal bisa membacanya. Tabel *CSV* berguna sebagai *format* perubahan data.

*f. Federated Engine*

Mesin penyimpanan ini semacam *proxy* ke *server* lain. Membuka koneksi *client* ke *server* lain dan mengeksekusi *query* melawan tabel yang ada, menyambungkan dan mengirim barisan yang dibutuhkan. Aslinya dipasarkan sebagai pesaing fitur yang mendukung dalam banyak *database server* seperti *Microsoft SQL Server* dan *Oracle*, tapi selalu ketat. Meskipun kelihatan lebih banyak fleksibilitas dan banyak trik, telah dibuktikan sumber banyak masalah. Kesuksesannya, *FederatedX* tersedia dalam *MariaDB.*

*g. Memory Engine*

*MySQL* memakai *memory engine* secara internal saat memproses *query* yang dibutuhkan tabel sementara untuk menahan hasil tingkat menengah. Jika hasil tingkat menengah menjadi terlalu besar untuk tabel *memory*, atau kolom *TEXT* atau *BLOB, MySQL* akan dikonversi ke dalam tabel *MyISAM* di *disk*.

*h. Merge Storage Engine*

*Merge engine* adalah variasi *MyISAM*. Tabel *merge* adalah kombinasi beberapa indentitas tabel *MyISAM* ke dalam tabel satu virtual. Ini bisa berguna ketika

memakai *MySQL*. Dalam *log* dan aplikasi data *warehousing*, tapi lebih diapresiasi bantuan partisi.

*i. NDB Cluster Engine*

*MySQL* memperoleh *NDB database* dari *Sony Ericsson* tahun 2003 dan membangun *NDB cluster storage engine* sebagai *interface* diantara *SQL* yang dipakai di *MySQL* dan protokol *NDB Native*. Kombinasi *MySQL server, NDB Cluster storage engine*, dan terdistribusi, tidak berbagi apapun, toleransi kesalahan, dan ketersediaan tinggi *database NDB* diketahui sebagai *MySQL cluster*.

## 2.9 Jenis – Jenis MySQL

Ada beberapa jenis *MySQL* seperti:

*1. Oracle*

Sejak kemampuan dan skalabilitas, kemampuan nyata, dan mudah dipakai sebagai *database* terpopuler di dunia, karakteristik membuat *MySQL* pilihan nomor satu untuk aplikasi *web* yang telah dibuktikan. Performa layanan *database* aplikasi *web* butuh bukti yang performa ekstrim untuk *read* keduanya (sederhana dan *query* kompleks) dan operasi *write*. Faktor performa lain adalah jenis sama performa ekstrim haruslah menunjukkan tidak masalah beban kerja (misal ratusan koneksi) atau volume data (*gigabyte* ke *terrabyte*). Kunci kekuatan untuk membuat performa tercepat seperti menggunakan *innodb*, sebuah *engine wide*

*storage* yang dibangun pada *MySQL*. *Innodb* membuktikan tingginya efisiensi *ACID* dan elemen arsitektur unik yang meyakini performa tinggi dan skalabilitas. Lalu ada mereplika *MySQL* dengan sederhana dan cepat membuat *multiple* replika database untuk diluar skala kapasitas dengan satuan permintaan. Berikutnya *MySQL* menawarkan spesialisasi *thread*/koneksi *cache* yang sangat cepat bila ada koneksi baru dan efisien mengakhiri koneksi yang ada pada permintaan. Koneksi *MySQL* selalu ada *thread* yang siap melayani permintaan pelanggan baru jadi tidak ada pengeluaran tambahan yang terbuang membangun koneksi dari dasar. Terakhir *MySQL* mengeksploitasi suplai memori pada *server* modern untuk menyampaikan yang merupakan pengecualian performa *database*. Tentu, utilitas *MySQL* data standar industri dan *index cache* untuk tetap informasi sering dalam memori untuk kecepatan akses [33].

2. *SQL Server*

Performa *SQL Server* terbaiknya, dengan bijak hanya meminta *metadata* tentang enkripsi untuk *query* disitu yang selalu pakai enkripsi. Menurut Stacia dan temannya maksud bahwa aplikasi untuk persentasi besar *query* selalu pakai enkripsi, *connection string* harusnya diperbolehkan dan *query* yang spesifik dengan aplikasi harus spesifik *sqlcommandcolumnencriptionsetting* sepertinya *disallowed*. Selain performa keamanan ada juga performa *high availability* atau ketersediaan tinggi lebih baik karena dasar *group availibility* memakai operasi *log transport* yang sama. Studi kasus pada performa tinggi *SQL Server* adalah

penyimpanan utama untuk *hyper-v file virtual*. Jika butuh tingkatan lebih performa lebih tinggi lagi, bisa konfigurasi penyimpanan kapasitas. *Sql server* bisa ambil keuntungan fitur ini karena infrastruktur mendukung selalu ada *group availability* dan selalu ada klaster kegagalan secara otomatis. Tentunya lebih hemat pengeluaran sang pemilik dibandingkan solusi penyimpanan yang lain [34].

3. *Microsoft Access*

*Microsoft Access* adalah sistem manajemen *database* relasional dari microsoft yang menggabungkan relasional *microsoft jet database engine* dengan antarmuka pengguna grafis dan pengembangan perangkat lunak. Menurut penelitian Youseff, *microsoft access* adalah anggota dari sistem *microsoft office*. Salah satu manfaat dari akses dari perspektif *programmer* adalah kompatibilitasnya relatif dengan *query SQL*. Tidak seperti *RDBMS* selesai, mesin *jet* tidak memiliki pemicu database dan prosedur yang tersimpan. Meskipun, ia menyediakan sintaks khusus yang memunkinkan menciptakan *query* dengan *parameter*, dengan cara yang terlihat seperti menciptakan prosedur yang tersimpan, tapi prosedur ini dibatasi satu pernyataan per prosedur. *Microsoft access* tidak memungkinkan bentuk mengandung kode yang dipicu perubahan yang dibuat ke meja yang mendasari, dan itu adalah umum untuk menggunakan *pass-through query* dan teknik lainnya di akses untuk menjalankan prosedur yang tersimpan di *RDBMS* yang mendukung ini. *Microsoft access* digunakan oleh usaha kecil, dalam departemen dari perusahaan besar, dan oleh *programmer* yang hobi membuat *ad hoc data*.

Beberapa pengembang aplikasi profesional menggunakan *access* untuk pengembangan aplikasi yang cepat, terutama untuk pembuatan prototipe dan aplikasi mandiri yang berfungsi sebagai alat untuk penjualan.

4. *IBM DB*

*DB2* adalah salah satu jalur *IBM* sistem manajemen *database* relasional yang berjalan pada mesin *server unix, windows,* atau *linux*. Pendapat Youseff, *DB2* dapat diberikan baik dari baris perintah atau antarmuka *GUI*. Baris perintah antarmuka membutuhkan lebih banyak pengetahuan tentang produk tetapi dapat lebih mudah ditulis dan otomatis. *GUI* adalah klien *java multi-platform* yang berisi berbagai tuntunan cocok untuk pengguna pemula. *Db2* mendukung kedua *SQL* dan *Xquery*. *DB2* memiliki implementasi asli penyimpanan data *XML*, dimana data *XML* disimpan sebagai *XML* untuk akses lebih cepat menggunakan *Xquery*. *DB2* juga mendukung intergrasi ke dalam *eclipse* dan lingkungan visual studi *NET* pengembangan terintegrasi. Sebuah fitur penting dari *DB2 DBMS* adalah pengolahan kesalahan dimana struktur bidang komunikasi *SQL* digunakan dalam program *DB2* untuk kembali informasi kesalahan ke program aplikasi setelah setiap panggilan *API* utnuk pernyataan *SQL*.

5. *MySQL*

*MySQL* gratis, *open-source, multithreaded,* dan *multi user*. Pendapat dari Youseff sistem manajemen *database SQL* yang memiliki lebih dari sepuluh juta instalasi.

Program dasar berjalan sebagai *server* menyediakan akses *multi user* untuk sejumlah *database*. *MySQL* termasuk *subset* luas *ANSI SQL 99*, serta ekstensi, dukungan *cross platform*, prosedur tersimpan, pemicu, kursor, pandangan di*update*, dan *X / Open XA* dukungan pemrosesan transaksi terdistribusi. Selain itu, mendukung dua tahap melakukan *engine*, mesin penyimpanan independen, dukungan *SSL*, *subquery caching*, replikasi dengan satu master per bantuan, banyak bantuan per induk, tertanam *database* perpustakan, dan *ACID* kepatuhan menggunakan *innodb* mesin klaster [35].

## 2.10 Evaluasi Tentang NoSQL Redis

*Redis* adalah *in-memory remote database* yang menawarkan performa tinggi, jawaban, dan data model yang unik untuk diproduksi ke *platform* untuk mengatasi masalah. Menurut Baron, *redis* itu data struktur *server*, dengan cepatnya di *in-memory key-value store* seperti *memcached*. Tapi *value* adalah struktur bukanlah *opaque* (buram). Banyak operasi yang dipakai di dalam *redis* seperti "add x to the set". Terlebih dengan adanya pemograman struktur data seperti *array* dan *hashes/dictionaries/maps/sets* [22][23].

*Redis* adalah tipe *database* yang tidak memiliki *table* dan bukan *database* yang didefinisikan atau relasi data dengan data lain di *redis*. Saat pernah memakai *memcached* sebelumnya, proses *Redis* hampir sama dengan *memcached* bahwa bisa menambah data sampai akhir *string* yang ada dengan *append*.

Menurut dokumentasi *memcached* bahwa *append* bisa dipakai mengatur daftar *item*. Bila ingin menghapus *item*-nya, *memcached* menjawab untuk memakai *blacklist* untuk menyembunyikan *item*, sekaligus menghindari *read/update/write* (atau *database query* dan *write memcached*). Di *redis* bisa memakai salah satu *list* atau *set* dan lalu menambahkan dan menghapus *item* langsung. Bila memakai *redis* dibandingkan *memcached* untuk saat ini dan masalah lain, bukan kode bahasa pemograman yang jadi lebih pendek, mudah dimengerti dan mudah ditangani, tapi lebih cepat (karena *redis* tidak butuh membaca *database* untuk *update* data). Ada banyak kasus lain dimana *redis* lebih efisien dan atau lebih mudah dipakai daripada relasi *database*.

Satu yang paling umum menyimpan data dengan jangka panjang *reporting* data. Untuk mengumpulkan agregat ini, *insert row* ke dalam *reporting table* dan nanti *scan* ke semua *row* untuk mengumpulkan agregatnya yang mana kemudian *update row* yang sudah ada di dalam *table* agregat. *Row* di *insert* karena kebanyakan *database*, memasukkan *row* adalah operasi lebih cepat [23].

Menurut Conor dan Patrick sebenarnya ada beberapa solusi *alternative* jika tidak ingin menggunakan *redis* namun tidak bisa menyalip campuran keunikan dari kemampuan yang *redis* sediakan. Kekayaan fitur yang lebih pada alternatif *database* seperti *mongodb* semacam lebih sumber daya intensif. Tabel berikut untuk menganalisa perbedaannya [24].

**Tabel 2.4** Tabel pembanding *nosql* alternatif selain *redis* [24].

| | *Redis* | *Memcached* | *Mongodb* |
|---|---|---|---|
| *In-memory* | √ | √ | |
| *Persistent* | √ | | √ |
| *Key-value store* | √ | √ | |
| *Supports more than strings* | √ | | √ |
| *Multithreaded* | | √ | √ |
| *Supports larger-than-memory dataset* | | | √ |
| *As fast as* | *Memory* | *Memory* | *Disk* |

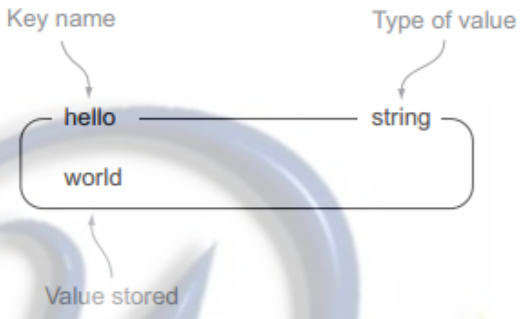
## 2.11 Fitur dan Cara Kerja Database NoSQL Redis

Josiah berpendapat *Redis* mempersilahkan *store keys* yang pemetaannya ke siapapun dari lima perbedaan tipe struktur data seperti *STRING, LIST, HASH*, dan *ZSET*. Tiap dari lima perbedaan struktur memiliki beberapa berbagi perintah (*DEL*, *TYPE*, *RENAME*, dan yang lainnya), sekaligus beberapa perintah yang hanya bisa dipakai satu atau dua struktur. Bisa dilihat pada tabel berikut ada lima struktur yang tersedia di *redis*.

**Tabel 2.5** Kelima struktur yang tersedia di *redis* [23].

| Tipe struktur | Apa isinya | Struktur kemampuan *read/write* |
|---|---|---|
| *STRING* | *String*, *integer*, or *float* nilai intinya | Mengoperasikan pada seluruh *string*, bagiannya, *increment/decrement interger* dan *float*. |
| *LIST* | Terhubung dengan daftar *string*. | *Push* atau *pop item* dari kedua akhir, memangkas berdasarkan *offset*, baca setiap *item* atau beberapa, menemukan atau menghapus *item* dengan nilai |
| *SET* | Tidak memesan kumpulan dari *string* yang unik. | Tambah, mengambil, atau menghapus individu *item*, cek keanggotaannya, silang-menyilang, menyatu, perbedaan, mengambil, *random item*. |
| *HASH* | Tidak memesan tabel kunci *hash* ke nilai | Tambah, mengambil, atau hapus *individual item*, mengambil keseluruhan *hash* |
| *ZSET (dipendekkan set)* | Memerintahkan pemetaan anggota string untuk skor *floating-point*, diperintahkan oleh skor. | Tambah, mengambil atau menghapus nilai individual, mengambil items berdasarkan pada jarak *score* atau nilai anggota |

Tipe struktur *string* di *redis* sama pada *string* yang ada di bahasa lain atau penyimpanan *key-value* lainnya. Pada umumnya, ketika menunjukkan diagram yang mewakili *key* dan *value*, diagram memiliki nama *key* dan jenis nilai bersama di atas kotak dengan nilai di dalam kotak itu.




**Gambar 2.3** Contoh *string*, *world*, disimpan di bawah *key*, *hello* [23].

Operasi tersedia ke *string* dimulai dengan apa yang tersedia di penyimpanan *key-value* yang lain. Bisa di nilai *GET*, nilai *SET*, dan nilai *DEL*. Setelah meng-*install* dan mencoba *redis* seperti yang digambarkan dengan *redis-cli*, bisa mencoba ke nilai *SET*, *GET*, dan *DEL* di *redis*, seperti yang terlihat di tabel dibawah ini dengan dasar arti fungsi yang tergambarkan di tabel berikut.

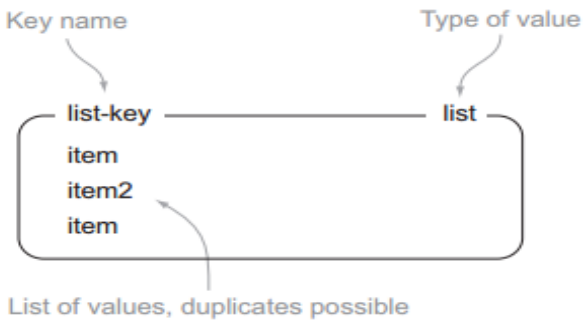
**Tabel 2.6** Perintah yang dipakai pada nilai *string* [23].

| Perintah | Apa yang dilakukan |
|---|---|
| *GET* | Mengambil data *stored* saat diberikan *key*-nya. |
| *SET* | Memasang nilai yang tersimpan saat diberikan *key*-nya. |
| *DEL* | Menghapus nilai yang tersimpan saat diberikan *key*-nya (bisa untuk semua tipe). |


**Gambar 2.4** Contoh yang menunjukkan perintah *SET*, *GET*, dan *DEL* di *redis* [23].

Menambahkan mampu untuk nilai *GET*, *SET*, dan *DEL STRING*, ada perintah lain untuk membaca dan menulis bagian *string* dan perintah yang membiarkan merawat *string* sebagai jumlah ke *increment/decrement* -nya.

Berikutnya akan membahas tentang tipe struktur *list* di *redis*. *List* di penyimpanan *redis* memerintahkan urutan *string*, dan seperti string, Mewakili sosok *list* sebagai *label box* dengan *list item* di dalamnya. Contoh *list* bisa dilihat di gambar berikut.

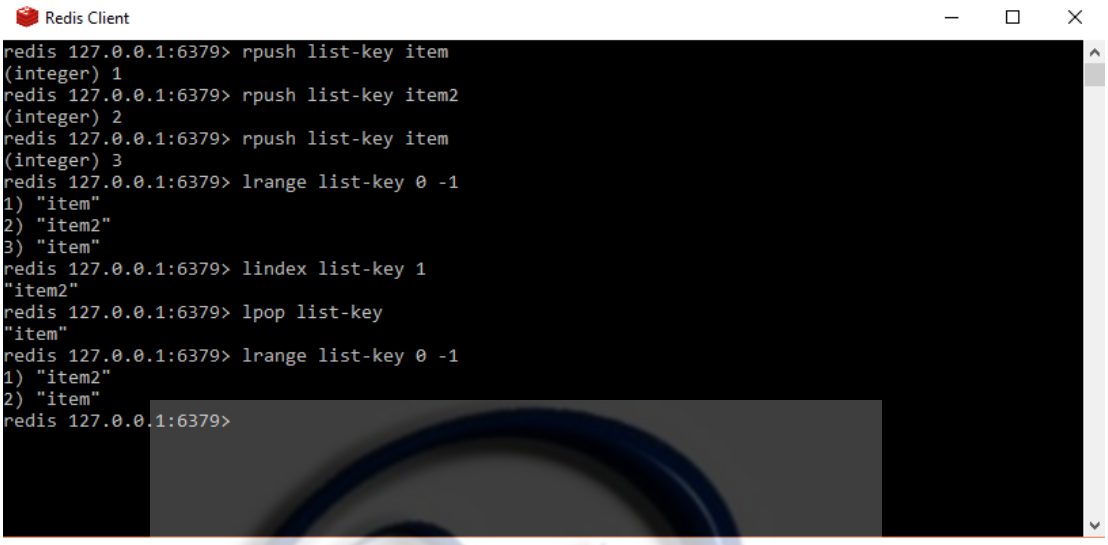



**Gambar 2.5** Contoh *list* dengan tiga *item* dibawah *list-key* [23].

Operasinya bisa dibentuk pada *list* yang jenisnya apa yang hampir sering ditemukan di bahasa pemograman yang ada. Saat masukkan *item* ke depan dan kembali ke *list* dengan *lpush/rpush*; menampilkan *item* dari depan dan belakang dan kembali ke daftar dengan *lpop/rpop*; Mengambil *item* yang posisi diberikan dengan *lindex*; dan Mengambil jarak *item* dengan *lrange*. Akan diperlihatkan deskripsi perintah yang akan dipakai [23].

**Tabel 2.7** Perintah yang ada dalam *list* [23].

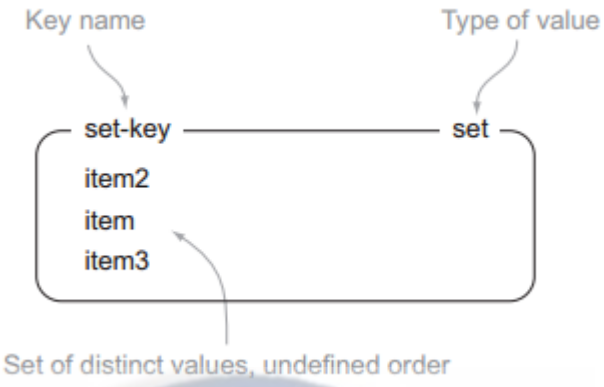
| Perintah | Apa yang dilakukan |
|---|---|
| *RPUSH* | Memasukkan (push) nilai kepada kanan akhir daftar. |
| *LRANGE* | Mengambil jarak nilai dari daftar |
| *LINDEX* | Mengambil *item* saat posisi diberikan di dalam daftar |
| *LPOP* | Muncul nilai dari kiri akhir daftar dan kembali. |




**Gambar 2.6** Contoh perintah *rpush, lrange, lindex, lpop* di *redis* [23].

Pada *rpush list-key item, rpush list-key item* dan *rpush list-key item2* menjelaskan tentang ketika memasukkan *item* kepada daftar, perintah kembali panjang arus daftarnya. Berikutnya *lrange list-key 0 -1* yang hasilnya ada 3 *item* terdaftar menjelaskan tentang bisa mengambil seluruh daftar dengan mengumpan jarak dari 0 untuk memulai *index* dan -1 untuk *index* terakhir. Berikutnya *lindex list-key 1* menjelaskan tentang mengambil *item* individu dari daftar dengan *lindex*. Terakhir *lrange list-key 0 -1* menjelaskan kemunculan *item* dari daftar membuat tak lagi tersedia [23].

Berikutnya di *redis* ada *set* yang sama ke *list* yang mana keduanya rantai *string* tapi tidak seperti *list*, *redis* set pakai *hash table* untuk tetap semua *string* unik. Penggambarannya seperti gambar di berikut ini,

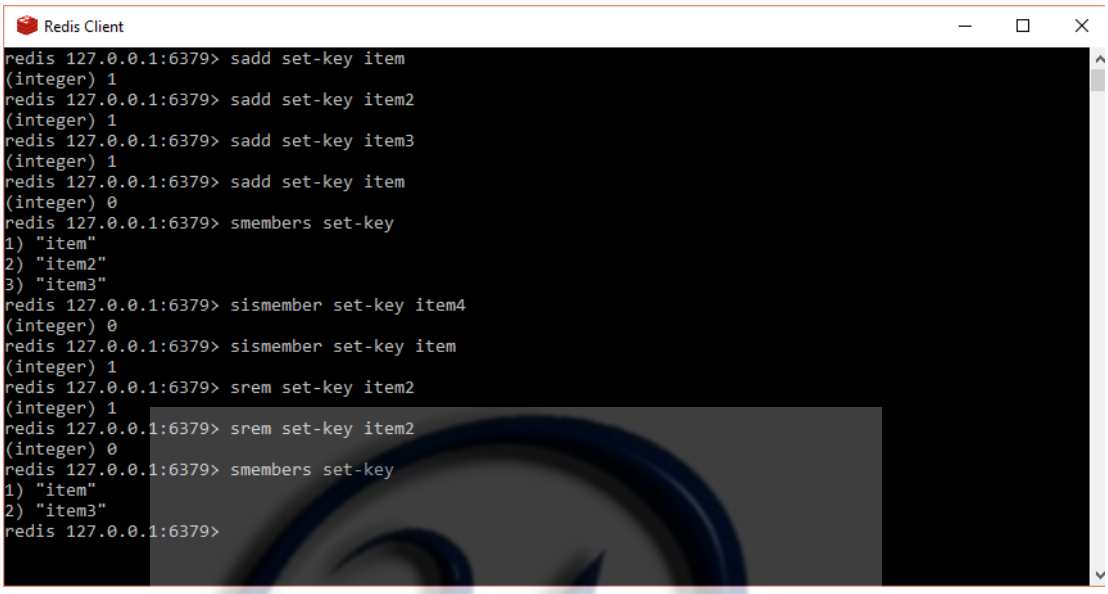



**Gambar 2.7** Contoh *set* dengan tiga *item* dibawah *set-key* [23].

Karena *redis set* tidak dipesan, maka tidak bisa memasukkan dan munculkan *item* dari akhir seperti proses tadi dengan *list*. Sebagai gantinya, perlunya *add* dan *remove item* yang ada di dalam *set* dengan cepat dengan *sismember*, atau mengambil semua *set* dengan *smembers* (ini bisa lambat untuk *set* yang besar, jadi waspadalah). Mengikuti sepanjang mendaftar di *redis client console* untuk dapat merasakan bagaimana *set* bisa jalan, dan ada tabel perintah yang mendeskripsikan di bawah ini [23].

**Tabel 2.8** Perintah dengan menggunakan nilai *set* [23].

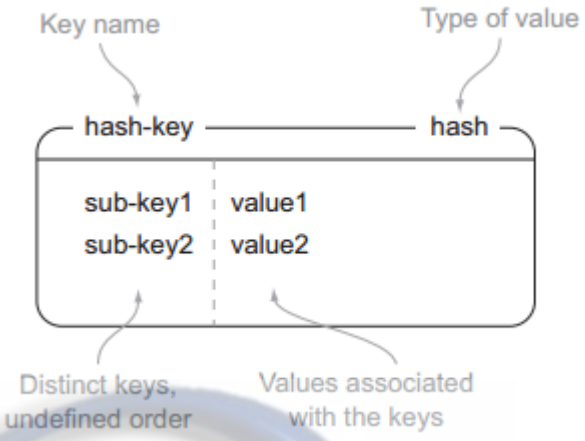
| Perintah | Apa yang dilakukan |
|---|---|
| *Sadd* | Menambah *item* ke *set* |
| *Smembers* | Kembali ke seluruh *set item* |
| *Sismember* | Cek jika *item* ada di dalam *set* |
| *Srem* | Hapus *item* dari *set*, jika masih ada |




**Gambar 2.8** Contoh perintah *sadd, smembers, sismember, srem* dan *smembers* [23].

Sepertinya mungkin berdasarkan pada *string* dan bagian *list*, *set* telah banyak satu lagi dipakai luar penambahan dan penghapusan *item*. Seperti biasa tiga dipakai operasi dengan *set* termasuk *intersection, union* dan perbedaan (*sinter, sunion*, dan *sdiff*, dengan penuh kehormatan).

Sedangkan *list* dan *set* di *redis* menahan urutan *item, redis hash* menyimpan pemetaan kunci ke nilai. Nilai yang disimpan di *hash* adalah sama seperti apa yang bisa disimpan seperti normalnya *string*. *String* sendiri, atau bila nilai mungkin ditafsirkan sebagai jumlah, bahwa nilai mungkin kenaikan atau pengurangan. Contoh gambar di bawah ini bisa menjelaskan agar dapat mudah dipahami maksud dari penggunaan *hash*.

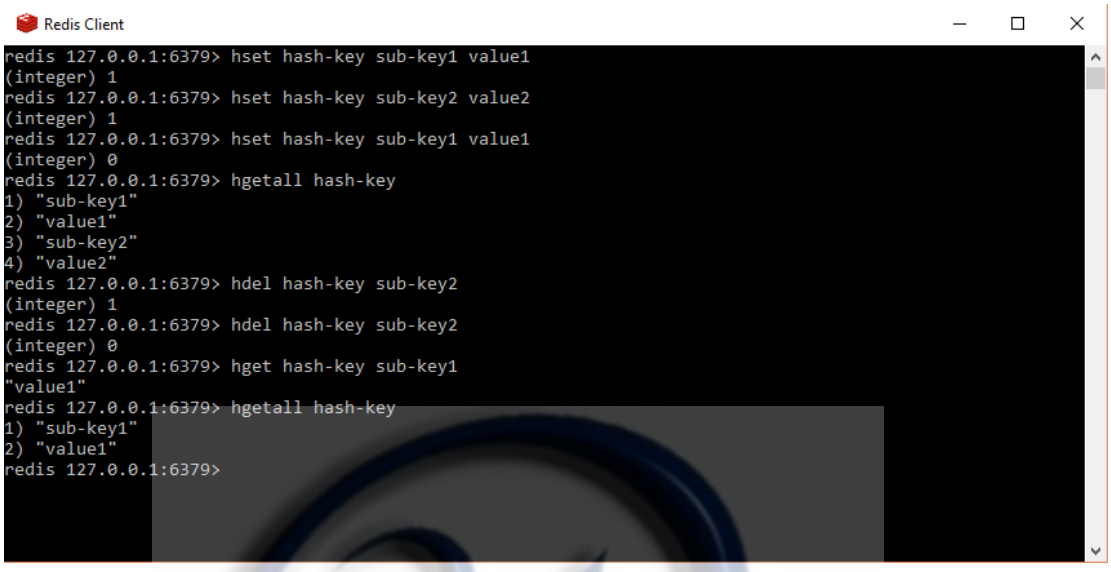



**Gambar 2.9** Contoh dari *hash* dengan dua *key value* dibawah *key hash* [23].

Beberapa perintah yang sama yang bisa ditampilkan pada *string*, Menampilkan pada nilai di dalam *hash* dengan sedikit perintah berbeda. Coba mengikuti tabel perintah *hash* di bawah ini.

**Tabel 2.9** Perintah dengan menggunakan nilai *hash* [23].

| Perintah | Apa yang bisa dilakukan |
|---|---|
| *HSET* | Menyimpan nilai saat kunci di *hash* |
| *HGET* | Mengambil nilai saat diberikan kunci *hash* |
| *HGETALL* | Mengambil seluruh *hash* |
| *HDEL* | Menghapus kunci dari *hash*, jika masih ada. |

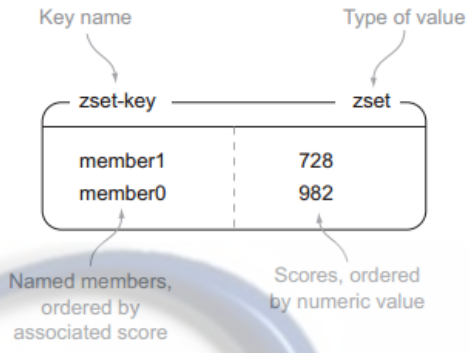



**Gambar 2.10** Contoh perintah praktikum di tabel *hash* [23].

Dari gambar di atas menjelaskan tentang *hset hash-key*…. Itu saat ketika menambahkan *item* ke *hash*, dapat nilai kembali yang dikatakan apakah *item* itu baru di *hash*. Berikutnya di *hgetall hash-key* menjelaskan tentang mengambil semua *item* di *hash*, yang mana dapat diterjemahkan ke dalam kamus di sisi hal *python*. Lalu *hdel hash-key sub-key2* mengenai ketika hapus *item* dari *hash*, perintah kembali apakah *item* tadi ada sebelum mencoba menghapusnya. Terakhir *hget hash-key sub-key1* menjelaskan bisa juga mengambil *field* individual dari *hash*.

Dan terakhir di *redis* yaitu *sorted set*, seperti *redis hash*, *zset* juga menaham tipe kunci dan nilai. Kunci (dinamakan *members*) adalah unik, dan nilai (dinamakan *scores*) adalah terbatas untuk jumlah poin yang sering berpindah-pindah. *Zset* memiliki *property* yang unik di *redis* mampu mengakses oleh *member* (seperti *hash*),

tapi *item* bisa juga diakses oleh perintah yang diurutkan dan nilai *score*-nya. Contoh *zset* dengan dua *item* di bawah ini.

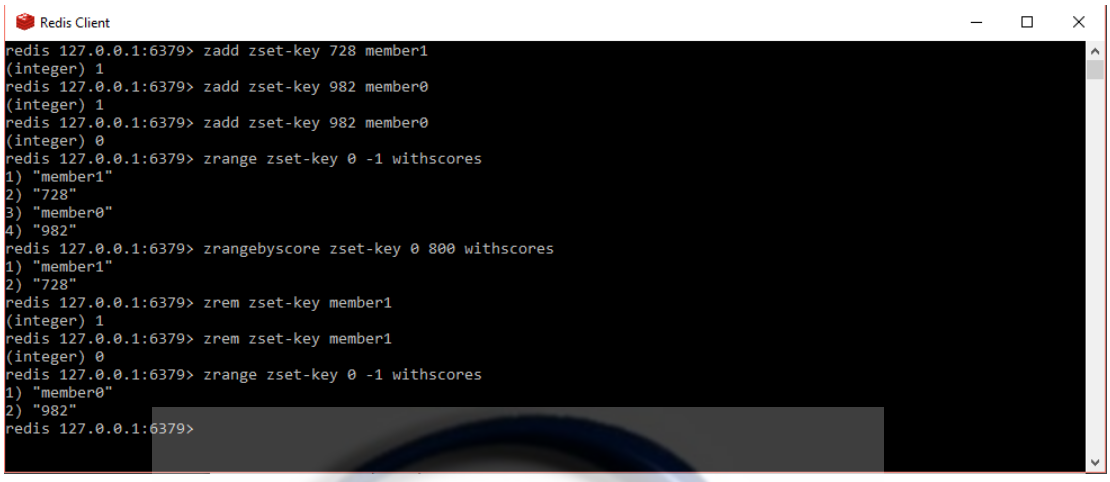



**Gambar 2.11** Contoh *zset* dengan dua *member/score* di bawah kunci *zset* [23].

Seperti pada kasus di atas, perlu membutuhkan menambah, menghapus dan mengambil *item* dari *zset*. Contoh 2.9 menawarkan menambah, menghapus dan mengambil perintah untuk *zset* sama untuk struktur lain, dan tabel 2.9 menggambarkan perintah yang akan dipakai.

**Tabel 2.10** Perintah yang dipakai dengan nilai *zset* [23].

| Perintah | Apa yang dilakukan |
|---|---|
| *Zadd* | Menambah anggota dengan memberikan *score* ke *zset* |
| *Zrange* | Mengambil *item* di *zset* dari posisi mereka di dalam perintah pengurutan. |
| *Zrangebyscore* | Mengambil *item* di *zset* berdasarkan pada jarak *score* |
| *Zrem* | Menghapus *item* dari *zset* jika ada |




**Gambar 2.12** Contoh perintah praktikum di *zset* [23].

Dari gambar praktikum di atas dapat dijelaskan dari *zadd zset-key*… ketika menambah *item* ke *zset*, perintah kembali ke jumlah *item* baru. Berikutnya *zrange zset-key*… bisa juga mengambil semua *item* di *zset*, yang mana dipesan oleh *score*, dan *score* kembali ke dalam *float* di *phyton*. Berikutnya *zrangebyscore zset-key*… bisa juga mengambil *subsequence* dari item berdasarkan *score*. Terakhir *zrange zset-key*…. Ketika perlu menghapus *item*, menemukan lagi jumlah *item* yang dihapus [23].

## 2.12 Studi Kasus Penelitian Aplikasi Sosial Media memakai Redis

Salah satu contoh ditampilkan pada *redis website* adalah *twitter clone* dikembangkan dengan *redis* dan *php*. Berdasarkan itu, dikembangkan versi sederhana *sosial media* berdasarkan *twitter* menggunakan *redis-py*: sebuah *client library phyton redis*. Implementasinya berdasarkan buku "redis in action" menurut karya Josiah Carlson. Tujuan utama memilih jenis aplikasi ini adalah men-*generate* setiap data untuk disumasilkan ke aplikasi nyata, ketika dibandingkan ke aplikasi *chat*. Selain

itu, domain ini lebih mudah dimodelkan di dalam relasi *database*. Oleh karena itu, saat menyimpan jumlah data yang sama di dalam *SQLite* agar menilai bagaimana *query* akan tampil ketika dibandingkan ke *redis*.

Di *redis*, memakai struktur data utama untuk menahan data jaringan sosial diikuti seperti:

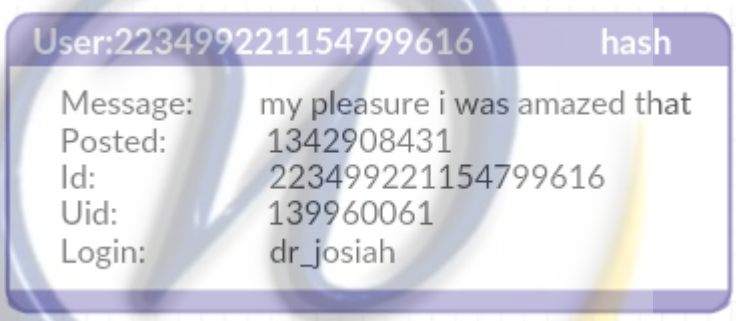
Informasi *user* (*login*, berapa banyak orang yang *follow*, berapa banyak *follower* yang mereka miliki, dan jumlah status pesan yang mereka *posting*) disimpan sebagai *hash*. `user: <id>, {followers: 0, following: 0, posts: 0, login: <username>}`

Penjelasan mengenai *script* di atas bahwa *user* yang disimpan sebagai *hash* akan mendapatkan *id* secara acak dari *redis*, berikutnya *followers*, *following* dan *post* semua itu tergantung pada *user* saat menggunakan aplikasinya, apabila memiliki *user* lain yang saling *follow* maka akan tersimpan di *database* dengan jumlah sesuai yang dimiliki *user*. Dalam penggambarannya bisa dilihat di bawah ini.


**Gambar 2.13** *Hash user table* [27].

Status pesan juga disimpan sebagai *hash*: `status: <id>, {uid: <user_id>, message: <message content>}`

Penjelasan *script* pada status hampir sama dengan *user*, bahwa status dalam bentuk *hash* akan mendapatkan *id* secara acak dari *redis* begitu juga *user_id*. *Message* tergantung dari *user* menggunakan aplikasi, apa yang ingin disampaikan lewat *post*ingan akan tersimpan ke *database*. Gambarannya bisa dilihat di bawah ini.




**Gambar 2.14** *Hash status table* [27].

Saat menyimpan seperti *zset* (semacam *set*) daftar *user* yang tiap *user follow*, dan *user* yang *follow* mereka. Untuk tetap pada daftar *follower* dan daftar orang disana yang *user following*, dan juga akan menyimpan *user id* dan *timestamp* di dalam *zset* juga, dengan anggota menjadi *user id*, dan *skor* menjadi *timestamp* ketika *user* di *follow* seperti *query* di bawah ini:

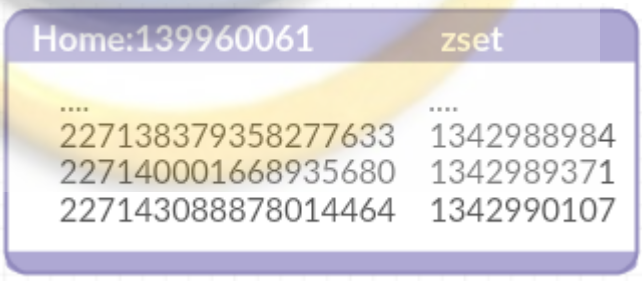
```
followers:<user_id>, [<follower_id> <timestamp>, <follower_id> <timestamp>, <follower_id> <timestamp> …]
following:<user_id>, [<following_id> <timestamp>, <following_id> <timestamp>, <following_id> <timestamp> …]
```

Untuk penggambarannya bisa dilihat di bawah ini.


**Gambar 2.15** *Followers* dan *following table* pada *zset* [27].

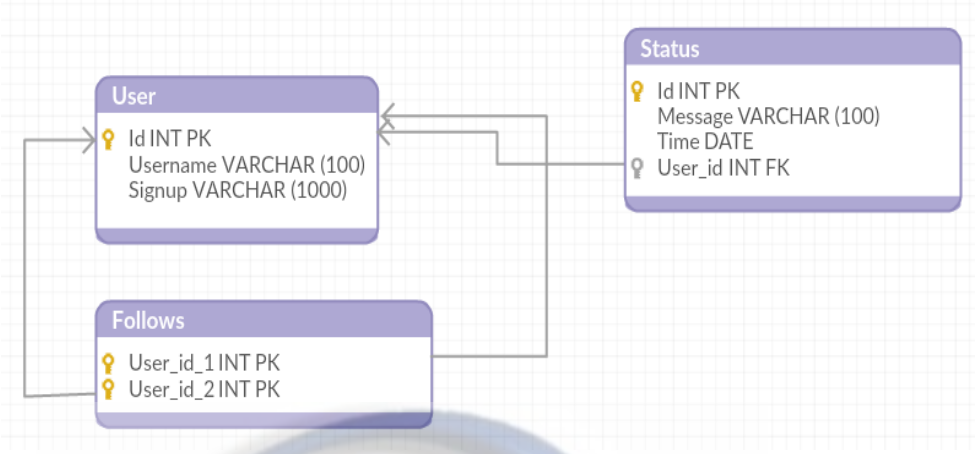
Akhirnya, *timeline*, paling penting fitur aplikasi, juga disimpan sebagai *zset*, agar tetap statusnya dipesan oleh *timestamp* (dipanggil *score* di dalam *redis*): `home:<user_id>, [<status_id> <status_timestamp>, <status_id> <Status_timestamp>, <status_id> <status_timestamp>, …]`

*Script* di atas menjelaskan pada tabel *home* menggunakan *zset* dengan *id* acak pada *redis*, baik itu *user_id*, *status_id*, *status_timestamp*. Penggambarannya seperti di bawah ini.




**Gambar 2.16** *Home table* dengan *zset* [27].

Dalam relasi *database* menggunakan *SQLite*, relasi modelnya perlu diikuti seperti:




**Gambar 2.17** Model relasi menggunakan *SQLite* [27].

Pada gambar di atas menjelaskan tentang tiga tabel *database SQLite* yang saling berelasi. Pada tabel *status foreign key user_id* berelasi ke tabel *user* dengan *id* sebagai *primary key*. Berikutnya tabel *follow* kedua atributnya *primary key* berelasi ke tabel *user* dengan *id* sebagai *primary key*.

Dalam aplikasi pada dasarnya dibuat lima metode:

**Tabel 2.11** Lima metode antara *redis* dan *SQLite* [27].

| | *Redis* | *SQLite* |
|---|---|---|
| *Create_user (username)* | Masukkan *user* baru sebagai *hash* | Masukkan *user* baru ke tabel *user* |
| *Post_status (user, message)* | Masukkan status baru sebagai *hash* dan tambah status *id* ke sejenis *set* setiap *follower user* yang | Masukkan barisan baru ke tabel *follow*. |

| | *Redis* | *SQLite* |
|---|---|---|
| | cocok. | |
| *Follow_user (user1, user2)* | Masukkan *user id* dari *user* 1 ke *follower* disusun *set user* 2 dan masukkan *user id* dari *user* 2 ke *following* disusun *set user* 1 | Masukkan barisan baru ke tabel *follow*. |
| *Retrieve_timeline(user)* | Mengambil *timeline* dari *user* yang sesuai dengan memilih semua status *id* dan mendapatkan isi status dari *hash* status. | Mengambil *timeline* dari *user* yang sesuai dengan menggabungkan tabel *user* dan status pada *user id*. |
| *Retrieve_followers(user)* | Mengambil informasi dasar dari semua *follower* dari *user* yang sesuai dengan mendapakan semua *follower* dari *follower* yang di-*set* dan mendapatkan informasi dari setiap *hash follower*. | Mendapatkan informasi dasar dari semua *follower* dari *user* yang sesuai dengan memilih semua *follower* dari tabel *follow* dan menggabungkan dengan tabel *user*. |

| | *Redis* | *SQLite* |
|---|---|---|
| *Retrieve_following(user)* | Mengambil informasi dasar dari semua *user* yang di *follow* oleh *user* yang sesuai, mengikuti dari *set* dan mendapatkan informasi dari setiap *hash user*. | Mengambil informasi dasar dari semua *follower* dari *user* yang sesuai dengan memilih semua *user* yang mem-*follow* dia dari tabel *follow* dan bergabung dengan tabel *user*. |

Metode yang menandakan sama untuk kedua versi aplikasi tersebut, menggunakan *redis* dan *SQLite*.

Dengan menghasilkan data yang sesuai untuk me-*register*-kan 1000 *user*, setiap *user* dengan 200 *follower* dan 100 pesan status. Dalam konteks ini, menghitung waktu dalam *miliseconds* untuk menjalankan *query* merespon ketiga metode tersebut:

**Tabel 2.12** Hasil akhir perbandingan *performance* kedua *database* [27].

| | *Redis* | *SQLite* |
|---|---|---|
| *Retrieve_timeline* | 0.36213 | 456.429 |
| *Retrieve_followers* | 0.05584 | 0.17909 |
| *Retrieve_following* | 0.04713 | 0.18737 |

Jumlah disini menunjukkan bahwa meski *redis* lebih lama menulis semua data ke *database*, *performance* ketika mengambil seperti data, khususnya dalam kasus *timeline*, secara signifikan lebih baik ketika dibandingkan ke *SQLite* [27].

## 2.13 Javascript Framework

Menurut *Microsoft* bahwa *javascript* tidak punya hubungan dengan *java*, walaupun begitu juga dengan bahasa *C#, C++*, dan banyak pemograman lainnya. *Javascript* berhubungan dengan *ECMAScript*. *ECMAScript* merupakan *standard script language* yang dikembangkan oleh kerjasama antara *netscape* dan *microsoft* dengan *widely used scripting language* bahwa dipakai di *web page* untuk mempengaruhi bagaimana keliahatan atau menunjukkan reaksi bagi pengguna.

*Javascript* adalah *untyped*, yang berarti bahwa ketika membuat sebauh *variabel*, tidak perlu tentukan jenisnya. Hal ini mungkin tampak seperti fitur karena mendeklarasikan *variabel* dengan *variabel*. Kata kunci dan menetapkan *string* ke *variabel*, dan kemudian, dapat menetapkan nomor telepon ke *variabel* yang sama. Namun, sulit menetapkan *microsoft intellisense* untuk bahasa *untyped*. Itu juga sulit untuk mempertahankan kode karena sulit mengetahui jenis *variabel*. Ini mungkin menyebabkan untuk mengembangkan ketidaksukaan langsung untuk bahasa, tapi bertahan dan bekerja dengan bahasa pemograman itu. Mengagumi pada kekuatan *javascript*, meskipun mungkin tidak ingin memakai *visual basic .NET* atau lainnya. Bisa menemukan semakin banyak waktu yang dihabiskan dengan *javascript*, lebih menghormati mengembangkan untuk bahasa ini [10].

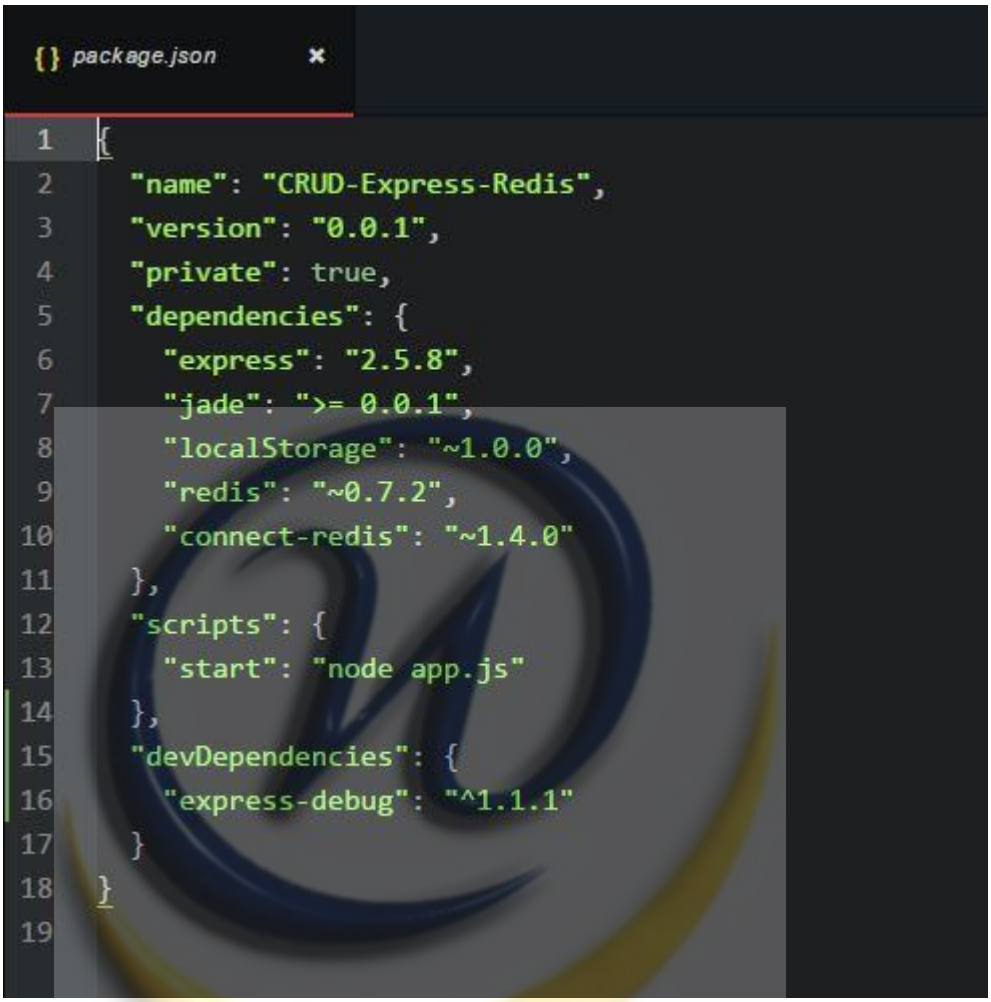
Selain *javascript* ada juga *NodeJs* dibuat tahun 2009 dan telah dikembangkan dan ditangani oleh sponsor *Joyent*. *NodeJs* menggunakan inti Google *open source* V8 *Javascript engine* [7].

## 2.14 JSON

*JSON* atau *JavaScript Object Notation* adalah berbasis teks standar terbuka yang kelas ringan dirancang untuk pertukaran data yang dapat dibaca manusia. Konvensi yang digunakan oleh *JSON* adalah diketahui *programer*, yang meliputi *C, C++, Java, Phyton* dan lainnya.

*Format JSON* ditentukan oleh Douglas Crockford. Ini dirancang untuk pertukaran data yang dapat dibaca manusia. *JSON* telah diperpanjang dari bahasa *scripting Javascript*. Perpanjangan dengan nama file *.json*. Jenis *JSON internet* media adalah aplikasi / *json*. *The uniform type identified* adalah *public .json* [11].

Mengikuti contoh menunjukkan bagaimana memakai *JSON* untuk menyimpan informasi seperti *script* pada gambar 2.18;




**Gambar 2.18** *Script json* yang diimplementasikan.

## 2.15 NodeJs

*NodeJs* bukanlah adalah *platform* bukan *server* seperti yang sudah dijelaskan sebelumnya di bagian *expressJs*. Dalam *nutshell*, *Nodejs* adalah *platform* perangkat lunak yang mengijinkan membuat *webserver* sendiri dan bangun *web* aplikasi terbaik. *Nodejs* bukanlah *webserver* sendiri, juga bukanlah bahasa. Itu mengandung *HTTP server library*, maksudnya tidak butuh menjalankan program *web server* terpisah seperti *apache* atau *IIS* (Internet Information Service) [7].

*Node.js* dinamakan *node* adalah lingkungan *server side Javascript* berdasarkan implementasi *Google* yang dinamakan mesin *V8*. *V8* dan *node* lebih diimplementasikan dalam *C* dan *C++*, fokus pada *performance* dan konsumsi *low memory*. Dimanapun *V8* mendukung *Javascript* dalam *browser* (lebih banyak ke *Google Chrome*), sasaran *Node* untuk mendukung proses panjang *server*. *Javascript* sangat cocok untuk pendekatan ini karena mendukung *callback*. Contohnya, ketika *browser* beban *document*, *user* mengklik tombol, atau permintaan *Ajax* terpenuhi, memicu *event callback*. Fungsional *Javascript* secara alami membuat benar benar mudah dibuat fungsi objek yang bisa didaftarkan sebagai penanganan *event*.

*Express* bentuk minimalis dan fleksibel pada *web apps* NodeJS yang membuktikan beberapa fitur untuk mengembangkan *web* dan aplikasi mobile. Dengan di fasilitasi pengembangan aplikasi *web* berdasarkan *node* yang sangat cepat. Beberapa fitur inti *framework ExpressJS* [39]:

a. Memungkinkan untuk mengatur *middleware* (mengeksekusi kode apapun dan mengatur permintaan dan respon objek) untuk menanggapi permintaan *HTTP*.

b. Mendefinisikan tabel *routing* yang digunakan untuk melakukan berbagai tindakan berdasarkan metode *HTTP* dan *URL*.
c. Memungkinkan untuk secara dinamis membuat halaman *HTML* berdasarkan lewat argumen untuk *template*.

**2.16 ExpressJs**

*Expressjs* adalah *javascript web framework* yang membuat lebih mudah membangun secara *custom*. Didesain dengan pola *framework* pada umumnya yang sudah terbiasa memakai seperti *sinatra* yang dipakai pada bahasa pemograman *ruby on rails* pasti lebih mudah memahami, lebih mendukung untuk kebutuhan aplikasi modern. Untuk menggunakan *express*, ada dua tipe pengembang yang direkomendasikan. Pertama, pengembang harus memiliki pengalaman *front-end development* dan menjadi *full-stack programmer* menggunakan *javascript*, bahasa yang sudah tahu. Kedua adalah pemgembang yang memiliki pengalaman bahasa lain dari *javascript* dan memperluas pengetahuannya ke *node.js*. selain itu, sudah terbiasa dengan *html, css*, dan *javascript* juga diharapkan, sekaligus pengalaman dengan membuat permintaan *ajax*. Yang lebih penting adalah rasa ingin tahu membangun aplikasi *web*.

*ExpressJs* sangat diperlukan oleh *NodeJS* dalam area pengembangan karena *ExpressJS* membuktikan transisi yang lebih mudah untuk mengembangkan *website* dengan memakai *NodeJS* sebagai lapisan tengah untuk mengatur *routes, server* dan berbagai fitur dengan sangat mudah [36].

## 2.17 ES6 ECMAScript 6

*ECMAScript* didefinisikan sebagai *European Computer Manufacturers Association* (ECMA). Secara lebih spesifik disebut *ECMAScript* atau *ECMA-262*. *ECMAScript Technical Committee* disebut *TC39*. *TC39* memiliki bi-bulan pertemuan tatap muka. Disamping standar definisi, anggota *TC39* membuat dan uji implementasi spesifikasi kandidat untuk memverifikasi yang benar dan kemungkinan terjadi membuat implementasi yang dapat dioperasikan. Dalam periode 2014 – 2015 anggota saat itu termasuk,

1. Brendan Eich (*Mozilla, Javascript inventor*),
2. Allen Wirfs-Brock (*Mozilla*),
3. Dave Herman (*Mozilla*),
4. Brandon Benvie (*Mozilla*),
5. Mark Miller (*Google*),
6. Alex Russell (*Google, Dojo Toolkit*),
7. Erik Arvidsson (*Google, Traceur*),
8. Domenic Denicola (*Google*),
9. Luke Hoban (*Microsoft*),
10. Yehuda Katz (*Tilde Inc, Ember.js*),
11. Rick Waldron (Boucoup, Jquery), dan lain sebagainya.

Perbedaan mendasar dari *ES5* dibandingkan dengan *ES6 (ES5 vs ES6)* adalah *ECMAScript 5* tidak menambahkan *syntax* baru sedangkan *ECMAScript 6* bisa. *ES6* ke belakang cocok dengan *ES5*, yang mana ke belakang cocok dengan *ES3*. Banyak fitur *ES6* membuktikan *syntactic sugar* (*syntactic sugar* adalah *syntax* dengan bahasa pemograman yang didesain untuk buat hal lebih mudah dibaca atau di ekspresikan) untuk lebih ringkas kodenya.


**Gambar 2.19** *Syntatic sugar* [20].

Satu tujuan *ES6* dan jauh untuk buat *javascript* adalah target lebih baik untuk *compiling* dari ke bahasa lainnya.

*Transpiler* adalah tipe *compiler* yang mengambil *source code* program yang ditulis di dalam satu bahasa pemograman sebagai input dan memproduksi *source code* secara ekuivalen di dalam bahasa pemograman lainnya. Singkatnya meng-*compile* terjemahan kode dari satu bahasa ke yang lain contoh dari *Java* ke *bytecode*.

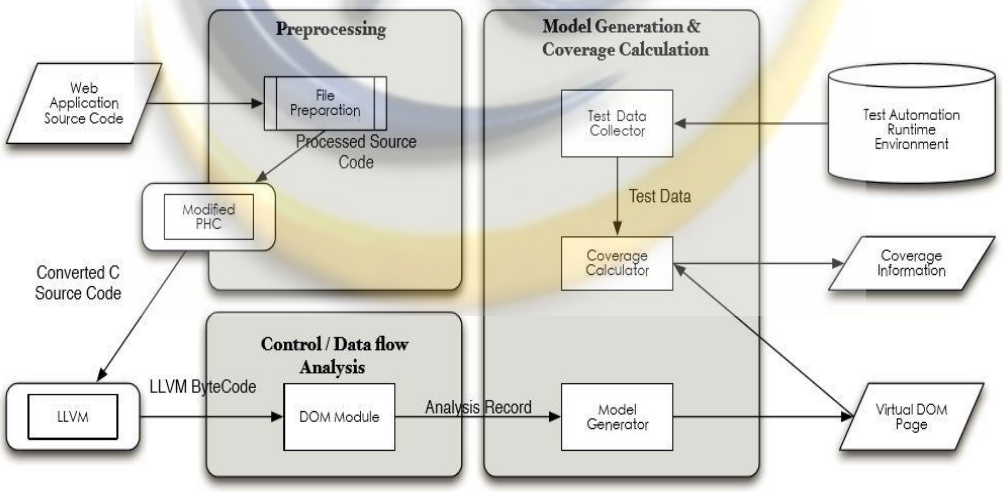
*Transpiler* menterjemahkan kode ke bahasa yang sama. Ada beberapa *transpiler* yang diterjemahkan kode *ES6* ke *ES5*.

*Traceur* – 64 % dari *Google*, menghasilkan *source map*. *Traceur* tidak bisa dipakai di *IE8* dan dibawahnya. Sedangkan *Babel* – 76% sasarannya menghasilkan kode *ES5* yang sedekat mungkin ke kode input *ES6*. Menghasilkan *source map* juga juga beberapa fitur tidak bisa dipakai dengan *IE10* dan dibawahnya. *Typescript* – 9% dari *microsoft*. Mengetik *superset javascript* yang *compile* ke *javascript* yang polos. Ke *browser* apapun, *host* apapun, *operating system* apapun, semuanya *open source* atau bisa dikomersilkan. Mendukung spesifikasi jenis pilihan untuk *variabel*, *function return value*, dan *function parameter*. *Typescript* juga memiliki tujuan untuk mendukung semua *ES6*. Juga menghasilkan *source map*. *Typescript* untuk menginstall cukup melakukan, `npm install -g typescript`. Untuk meng-*compile*-nya `tsc some-file.ts` [20].

**2.18 Virtual DOM**

Bahwa *V-Dom* dasarnya model eksekusi pada kedua sisi *server* dan *client*. Biasanya, tujuan perbedaan jalan eksekusi pada *server side* adalah untuk menyusun halaman *client* dalam perbedaan. Perbedaan dikumpulkan dan dilambangkan di *V-Dom*. Oleh karena itu, meliputi objek *DOM* menyiratkan meliputi jalur pada *server* side. Di sisi lain, setiap objek *DOM* bahwa pengguna *client* bisa berinteraksi dengan diwakili *V-Dom*, meliputi objek dalam *V-DOM* juga berarti mencakup semua

kemungkinan interaksi sisi *client*. Jika objek *DOM* diterjemahkan dalam uji coba, maka objek yang ditampilkan. Karenanya mendefinisikan tampilan cakupan *V-DOM* sebagai rasio dari benda yang ditampilkan oleh percobaan untuk semua benda di *V-DOM*. Di sisi *client*, pengguna bisa berinteraksi dengan aplikasi *web* dengan beroperasi pada kontrol pengguna seperti tombol atau mengetik teks kotak. Tindakan ini bisa memicu peristiwa yang berpotensi mengubah keadaan eksekusi. Sering disebut objek *DOM* yang telah melekat pada *event listener*. Jika spesifik *event* dari sebuah objek interaktif dipicu dalam uji coba, maka *event coverage*. Oleh karena itu, V-DOM *event coverage* adalah rasio jumlah peristiwa dipicu pada semua peristiwa yang ada di *V-DOM*. Mengukur kelengkapan tes dalam hal interaksi pengguna.




**Gambar 2.20** Arsitektur *V-DOM*. Kotak Abu-abu adalah tiga langkah utama. Yang disekitarnya berwarna putih kotak sisi melengkung dengan kotak kecil di dalamnya adalah alat yang dipakai. Dan kotak jajargenjang adalah data menengah atau hasil analisis [40].

Seperti yang ditunjukkan pada Gambar 2.20, proses arsitektur *V-DOM* terdiri dari tiga langkah utama:

*a. Preprocessing.*

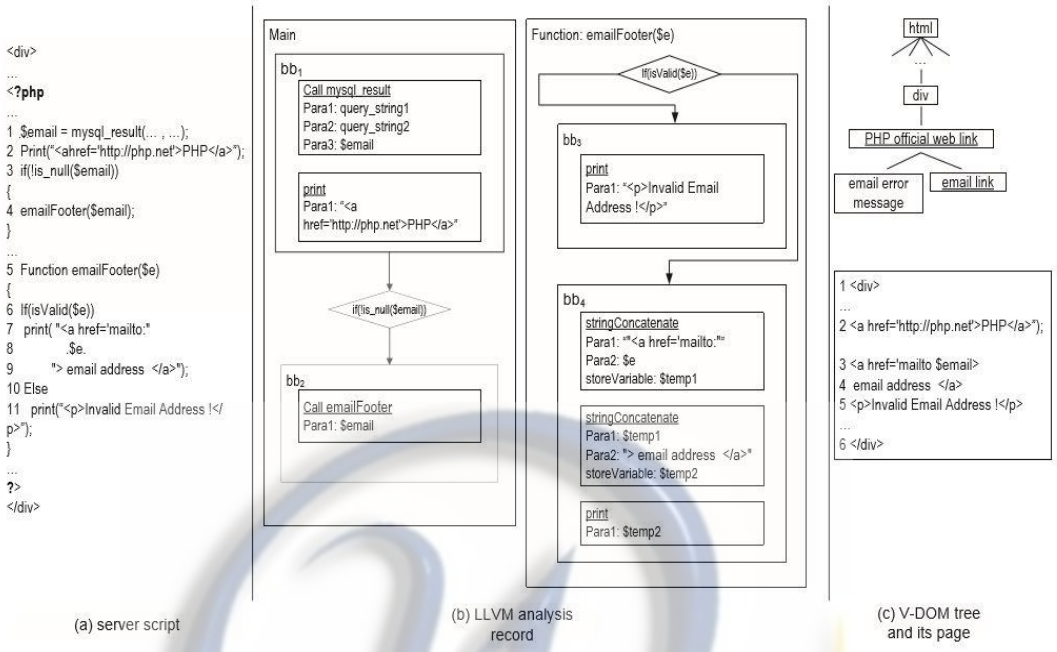
Pada langkah ini, aplikasi *web* disiapkan untuk analisis nanti statis. Terutama, inklusi naskah ditangani oleh termasuk isi dari skrip sasaran. *Script Server* dihasilkan kemudian diterjemahkan ke kode C.

*b. Control-flow* dan *Dataflow Analysis*.

Pada langkah ini, kontrol aliran statis dan *data-flow* analisis terbentuk pada kode C diterjemahkan untuk mengidentifikasi dan mengekstrak semua objek *DOM* yang mungkin dipancarkan oleh *script*. *Output* dalam langkah ini adalah catatan analisis *file*.

*c. Model Generation* dan Cakupan Komputasi.

Pada langkah ini, alat membuat *V-DOM* berdasarkan hasil analisis program pada langkah sebelumnya. Selama tes eksekusi kasus, data yang *runtime* yang terkait dengan kriteria cakupan dikumpulkan.




**Gambar 2.21** Contoh Konstruksi *V-DOM* [40].

Pada Gambar 2.21 menunjukkan contoh konstruksi *V-DOM*. *Script PHP* adalah versi penyederhanaan dari cuplikan di *timeclock* untuk *output footer* halaman. Itu output link ke situs resmi bahasa *PHP*. Dan *link* alamat *email* juga disediakan tepat setelah *link url* jika alamat tersebut valid. Alamat *email* diambil oleh *query database*. Jika nilai *string non-null* dikembalikan oleh *query*, suatu fungsi didefinisikan dipanggil untuk *output* informasi *email* (baris 3-4). Secara teori alamat *email valid*, itu dicetak (garis 7-9). Pesan peringatan dipancarkan sebaliknya. Pada tahap analisis *LLVM* sebelumnya, analisis *log record* yang dihasilkan, yang berisi *CFG* dan data ketergantungan informasi. *CFG* dari dua fungsi yang menunjukkan dalam gambar (b). Menurut Algoritma pada gambar di atas, *CFG* dari fungsi utama dilalui dalam urutan topologi. Dengan demikian, di halaman hasil pada gambar (c), pertama dua baris dari

halaman *client* yang dipancarkan sebagai dalam eksekusi nyata. Setelah mencapai dua fungsi pada baris 4, algoritma terus menganalisis fungsi *emailFooter()*. Dalam konteks ini, formal parameter $ bertekad untuk memiliki nilai simbolis "$ email" karena tidak bisa menentukan nilai kembali dari *query database*. Dalam fungsi, yang benar dan cabang – cabang palsu dilalui satu demi satu, yang mengarah ke *hyper-link* di baris 3-5 di (c) yang dipancarkan samping satu sama lain, keduanya termasuk dalam wadah yang sama. Bahwa algoritma mengevaluasi penggabungan *string* di baris 7-9 di (a) untuk memperoleh garis 3-4 di (c).

Dalam *Google Chrome*, implementasi *DOM* sangat jelas seperti pada *panel timeline* untuk merekam dan menganalisis semua aktifitas dalam aplikasi untuk memulai investigasi masalah *performance* dalam aplikasi saat berjalan. *Timeline Google Chrome* juga dipakai untuk merekam *tumpukan kode Javascript* juga merekam *performance* aplikasi yang dibangun dengan *javascript* [40].

## 2.19 Review Penelitian Studi Kasus

Di sini akan menampilkan laporan hasil kinerja dari beberapa peneliti di seluruh universitas ataupun perusahaan yang menyangkut tentang topik Tugas Akhir penulis. Ada beberapa daftar yang penulis ambil dari beberapa sumber dari hasil penelitian mereka, berikut daftarnya:

**Tabel 2.13** Daftar Referensi penelitian tentang *NoSQL & MySQL*.

| Nama Peneliti | Nama Jurnal | Tahun Penelitian | Penerbit |
|---|---|---|---|
| Robin Hect dan Stefan Jablonski | *NoSQL evaluation* | 2011 | IEEE |
| Lior Okman, Nurit Gal Oz, Yaron Gonen, Ehud Gudes, Jenny Abramov | *Security issues in NoSQL databases* | 2011 | IEEE |
| Jing Han, Haihong E, Guan Le, Jian Du | *Survey on NoSQL database* | 2011 | IEEE |
| Guoxi Wang, Jianfeng Tang | *The nosql principles and basic application of cassandra model* | 2012 | IEEE |
| Karamjit Kaur, Rinkle Rani | *Modeling and querying data in nosql databases* | 2013 | IEEE |
| Wumuti naheman, jianxin wei | *Review of nosql databases and performance testing on hbase* | 2013 | IEEE |
| Martin Fotache, Dragos Cogean | *Nosql and sql databases for mobile applications. Case study mongodb versus* | 2013 | IEEE |

| Nama Peneliti | Nama Jurnal | Tahun Penelitian | Penerbit |
|---|---|---|---|
| | *postgreSQL* | | |
| Yovita Tunardi, Rita Layona | *Nosql technology in android based mobile chat application using mongodb* | 2014 | IEEE |
| Chia Ping Tsai, Hung Chang Hsiao | *Streaming in nosql* | 2014 | IEEE |
| Ebrahim Sahafizadeh, Mohammad Ali Nematbakhsh | *A survey on security issues in big data and nosql* | 2015 | IEEE |
| Jagdev Bhogal, Imran Choksi | *Handling big data using nosql* | 2015 | IEEE |
| Maria Teresa Gonzalez Aparicio, Muhammad Younas, Javier Tuya, Ruben Casado | *A new model for testing CRUD operations in a nosql database* | 2016 | IEEE |
| Naveen Garg, Dr.Sanjay Singla, Dr.Surender Jangra | *Challenges and techniques for testing of big data* | 2016 | IEEE |
| Aryan Bansel, Horacio Gonzalez Velez, Adriana E Chis | *Cloud based nosql data migration* | 2016 | IEEE |
| Sergio Miranda Freire, | *Comparing the* | 2016 | IEEE |

| Nama Peneliti | Nama Jurnal | Tahun Penelitian | Penerbit |
|---|---|---|---|
| Douglas Teodoro, Fang Wei Kleiner, Erik Sundvaii, Daniel Karlsson, Patrick Lambrix | *performance of nosql approaches for managing archetype based electronic health record data* | | |
| Ying Ti Liao, Jiazheng Zhou, Chia Hung Lu, Shih Chang Chen, Ching Hsien Hsu, Wenguang Cheng, Mon Fong Jiang, Yeh Ching Chung | *Data adapter for querying and transformation between SQL and nosql database* | 2016 | IEEE |