BAB 2
TINJAUAN PUSTAKA

2.1. Landasan Teori

Pada bagian ini dijelaskan teori-teori yang menunjang terlaksananya pembangunan extension id generator dinamis pada postgresql.

2.1.1. Extension

Extension memungkinkan perluasan fungsionalitas dan konten khusus di luar aplikasi dan membuatnya tersedia bagi pengguna saat berinteraksi dengan aplikasi.

Tujuan kenapa aplikasi mendukung extension adalah:

1. Memperbolehkan pengembang pihak ketiga untuk meningkatkan dan mengembangkan kemampuan aplikasi.
2. Untuk mendukung kemudahan dalam penambahan fitur baru.
3. Untuk memperkecil ukuran aplikasi.
4. Untuk memisahkan kode sumber dari aplikasi dikarenakan lisensi perangkat lunak yang tidak sesuai.[8]

2.1.2. PostgreSQL Extension

Pada PostgreSQL sudah banyak terdapat fungsi, operator, tipe data dan agregasi namun terkadang pengguna masih perlu untuk membuat fungsinya sendiri agar kebutuhannya dapat terpenuhi dalam bentuk extension.[8]

2.1.2.1. Bagaimana Extensibility Bekerja

PostgreSQL dapat di *extend* karena operasinya *catalog-driven*. Informasi basis data mengenai tabel, kolom dan informasi lainnya disimoan dalam sistem katalog. (beberapa menyebutnya kamus data.) Katalog muncul kepada pengguna sebagai tabel pada umumnya namun DBMS (*Database Management System*) menyimpan pembukuan internalnya di dalam. Satu perbedaan antara PostgreSQL dan basis data relasi standar adalah PostgreSQL menyimpan lebih banyak informasi di dalam katalognya, tidak hanya informasi mengenai tabel dan kolom,

tetapi juga informasi mengenai tipe data, fungsi, metode akses dan sebagainya. Tabel ini dapat

dimodifikasi oleh pengguna, dan karena PostgreSQL mendasarkan operasinya pada tabel ini maka PostgreSQL dapat dikembangkan oleh pengguna. Sebagai perbandingan, sistem basis data konvensional hanya bisa dikembangkan dengan mengubah prosedur *hardcode* dalam kode sumber atau memuat modul khusus yang ditulis oleh vendor DBMS.

PostgreSQL dapat memasukkan kode yang ditulis pengguna ke dalam sistem melalui *dynamic loading*. Artinya pengguna dapat menentukan file kode objek (misalkan: *shared library*) yang mengimplementasikan tipe baru atau fungsi dan PostgreSQL akan memuatnya sesuai kebutuhan. PostgreSQL menyediakan empat bentuk *user-defined functions*:

1. *Query language functions* (fungsi yang ditulis dalam SQL).
2. *Prosedural language functions* (contoh PL/pgSQL atau PL/Tcl).
3. *Internal functions*.
4. C-*Language functions*.[8]

2.1.2.2. Procedural Language Functions

PostgreSQL memperbolehkan *user-defined functions* untuk ditulis dengan bahasa selain SQL dan C. Bahasa-bahasa lain ini secara umum disebut dengan istilah *prosedural language* (PL).

Fungsi yang ditulis menggunakan *procedural language* tidak bisa langsung di baca oleh basis data PostreSQL. Namun tugas dapat diteruskan ke *handler* khusus yang mengetahui detail bahasa. *Handler* yang digunakan akan melakukan semua pekerjaan *parsing*, analisis *sintaks*, eksekusi dan sebagainya atau bisa juga sebagai "perekat" antara PostgreSQL dan implementasi yang ada dari bahasa pemrograman yang digunakan. *Handler* merupakan bahasa C yang dikompilasi menjadi *shared object* dan dimuat sesuai permintaan, sama seperti fungsi yang ditulis pada bahasa C lainnya.

Saat ini terdapat empat *prosedural language* yang tersedia dalam paket distribusi PostgreSQL:

1. PL/pgSQL
2. PL/Tcl

3. PL/Perl
4. PL/Python[8]

2.1.2.3. PLGO (Procedural Language for GO)

PLGO adalah sebuah tool yang dapat digunakan untuk membuat PostgreSQL extension dengan menggunakan bahasa GO. PLGO akan membungkus kode dalam bahasa GO agar dapat dibaca dan dijalankan fungsinya oleh PostgreSQL.

2.1.3. Basis Data

Basis Data merupakan suatu kumpulan data terhubung yang disimpan secara bersama-sama pada suatu media, yang diorganisasikan berdasarkan skema atau struktur tertentu dan dengan perangkat lunak untuk melakukan manipulasi untuk kegunaan tertentu. Basis data bisa diartikan juga sebagai sekumpulan data yang disusun dalam bentuk tabal yang saling berelasi maupun berdiri sendiri[9].

2.1.4. Indeks (ID)

Nomor indeks (ID) digunakan untuk menunjukkan suatu data item (data item yang menjadi file, direktori, *record* dalam basis data, objek dalam pemrograman berbasis objek, lokasi pada memori atau perangkat fisik, atau sejenisnya) yang didefinisikan dalam konteks yang spesifik[10].

2.1.5. Akuntansi

Akuntansi adalah suatu disiplin yang menyediakan informasi penting sehingga memungkinkan adanya pelaksanaan dan penilaian jalannya perusahaan secara efisien. Akuntansi didefinisikan sebagai proses mengidentifikasikan, mengukur dan melaporkan informasi ekonomi untuk memungkinkan adanya penilaian dan keputusan yang jelas dan tegas bagi mereka yang menggunakan informasi tersebut.

Terdapat beberapa manfaat yang dirasa dengan adanya informasi akuntansi:

1. Sebagai alat ukur suatu keberhasilan ataupun kegagalan usaha dengan cara menilai harta ataupun hutang yang ada di perusahaan. Nilai dari uang dicatat dan dilaporkan menggunakan proses akuntansi. Sehingga

akuntansi memang merupakan *language of business*. Apa pun kegiatannya apabila terlibat dalam kegiatan usaha maka akan terasa manfaat dengan adanya ilmu akuntansi.

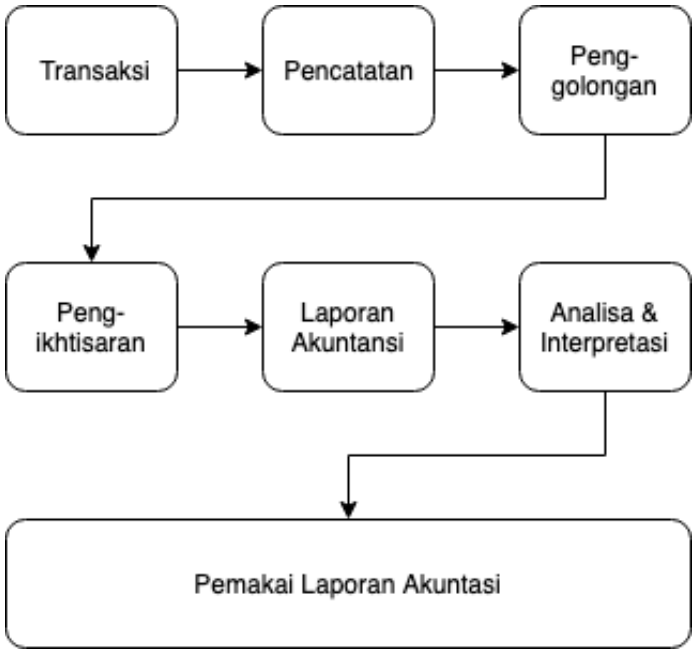
2. Akuntansi juga merupakan penghasil informasi yang dibutuhkan oleh pihak manajemen untuk melakukan perencanaan, mengeksekusi perencanaan dan melakukan kontrol aktivitas operasional usaha.
3. Menyajikan informasi ekonomi dari suatu kesatuan ekonomi kepada pihak-pihak yang berkepentingan. Informasi ekonomi yang dihasilkan oleh akuntansi berguna bagi pihak di dalam perusahaan itu sendiri maupun pihak-pihak dari luar perusahaan.[11]

2.1.5.1. Pembukuan dan Akuntansi

Pembukuan dan akuntansi saling berhubungan dan tidak ada pemisah yang tegas dan diterima secara umum. Pada umumnya pembukuan adalah pencatatan data perusahaan dengan suatu cara tertentu. Seseorang pemegang buku mungkin bertanggung jawab atas semua pencatatan dalam perusahaan atau hanya sebagian kecil saja dari kegiatan pencatatan dalam perusahaan tersebut.

Akuntansi terutama berhubungan dengan perencanaan sistem pencatatan, penyusunan laporan berdasarkan data yang telah dicatat dan penafsiran atas laporan-laporan tersebut. Akuntan akan memimpin dan mengawasi pekerjaan seorang pemegang buku.[11]

2.1.5.2. Proses Akuntansi



Gambar 2.1 Alur proses akuntansi.

Tahapan pencatatan:

1. Pembuatan atau penerimaan bukti untuk transaksi-transaksi yang terjadi dalam perusahaan.
2. Pencatatan bukti transaksi ke dalam Buku Harian (Jurnal) dan sekaligus menggolongkan transaksi tersebut ke dalam nomor perkiraan.
3. Pemindahan buku (*Posting*) dari buku harian ke perkiraan yang bersangkutan di buku besar (*Ledger*).

Tahap pengikhtisaran:

1. Pembuatan Neraca Saldo (*Trial Balance*) dari perkiraan-perkiraan di buku besar.
2. Pembuatan Neraca Lajur (*Worksheet*) dan Jurnal penyesuaian (*Adjustment Entries*).

3. Penyusunan Laporan Keuangan (*Income Statement/ Statement of Comprehensive Income, Statement of Financial Position/ Balance Sheet, Statement of Change in Equity/Capital Statement*).
4. Pembuatan Jurnal Penutupan (*Closing Entries*).
5. Pembuatan Neraca Saldo Penutupan (*Post Closing Trial Balance*).
6. Pembuatan Jurnal Balik (*Reversing Entries*).[11]

2.1.5.3. Transaksi

Transaksi keuangan tidak hanya menyangkut jual beli yang dilaksanakan ketika perusahaan mengeluarkan atau menerima uang tunai. Transaksi Keuangan yaitu semua kejadian yang menyangkut unit organisasi yang dapat diukur dalam satuan uang dan berpengaruh terhadap kekayaan organisasi (perusahaan). Ketika pemilik menyerahkan kendaraan atau rumah yang dapat digunakan untuk operasional dan menjadi hak milik perusahaan, kejadian tersebut termasuk transaksi keuangan, yaitu transaksi modal.

Setiap kegiatan transaksi harus disertai dengan alat bukti yang menyatakan bahwa telah terjadi suatu transaksi di mana bukti transaksi merupakan sumber di mana kita mengetahui apa saja yang telah terjadi di perusahaan dan sebagai bahan untuk melanjutkan tahapan-tahapan dalam proses akuntansi.

Suatu transaksi tersebut dapat dikatakan bila:

1. Menggunakan dokumen sesuai peruntukannya.
2. Diisi, di tanda tangani dan di validasi oleh pemegang otoritas, baik pembuat dan penerima.

Macam-macam bukti transaksi:

1. Kwitansi
2. Nota Kontan
3. Nota Kredit
4. Nota Debit
5. Faktur
6. Memo Internal

Transaksi memerlukan nomor unik untuk membedakan satu transaksi dengan transaksi lainnya.[11]

2.1.5.4. Chart of Account (COA)

Chart of Account (COA) atau bagan perkiraan adalah daftar yang memuat mengenai keseluruhan kode dan nama akun. Kode dan nama akun yang ada digunakan untuk mengklasifikasikan setiap transaksi bisnis yang terjadi.

Contoh penomoran COA:

1. Aset
    1.1 Kas
    1.2 Piutang Usaha
    1.3 Perlengkapan kantor
    1.4 Asuransi yang dibayarkan di muka
    1.5 Peralatan kantor
    1.6 Perabot kantor
2. Utang
    1.1 utang
    1.2 sewa diterima di muka
3. Ekuitas Pemilik
    3.1. Modal
    3.2. *Prive*
4. Pendapatan
    4.1 Pendapatan usaha
    4.2 Pendapatan sewa
    4.3 Pendapatan bunga
5. Beban
    5.1. Beban gaji
    5.2. Beban iklan
    5.3. Beban sewa kantor
    5.4. Beban utilitas
    5.5. Beban rupa-rupa

Bentuk baku dalam penyusunan COA dan yang diterapkan oleh kebanyakan perusahaan adalah bahwa pengelompokan kode (nomor) 1 selalu dimulai dari akun-akun aset, lalu diikuti dengan akun-akun dari kelompok utang, ekuitas, pendapatan dan beban[12].

Bentuk kode yang digunakan untuk COA dapat berupa angka, huruf, simbol atau gabungan ketiganya. Pada umumnya pemberian kode dilakukan untuk memberi simbol atau tanda terhadap klasifikasi yang telah dibuat sebelumnya. Kode yang tepat akan memberikan identifikasi yang unik dan ringkas yang dapat menggambarkan suatu entitas, kejadian, transaksi, maka akan memudahkan memasukkan data ke dalam suatu sistem, pada saat memanggil data dan mengolah data untuk menjadi sebuah bentuk yang diinginkan[13].

2.1.6. Tipe Data JSON

Tipe data JSON digunakan untuk menyimpan data JSON (*JavaScript Object Notation*), seperti yang didefinisikan pada RFC 7159. Data semacam itu dapat disimpan dengan format *text*, namun tipe data JSON memiliki kelebihan dalam penggunaannya karena data yang disimpan valid sesuai dengan aturan JSON.

Terdapat dua jenis JSON, json dan jsonb. Keduanya memiliki set nilai yang hampir identik sebagai masukan. Perbedaannya adalah dalam segi efisiensi. Tipe data json menyimpan salinan yang tepat dari masukan teks, yang fungsi pemrosesannya harus berulang pada setiap eksekusi, sementara jsonb disimpan dalam format biner terurai yang membuatnya sedikit lambar untuk di masukan karena terjadi penambahan *overhead* konversi, tetapi secara signifikan lebih cepat untuk di proses, karena tidak perlu reparasi. Jsonb juga mendukung pengindeksan yang menjadi keuntungan signifikan[8].

2.2. Konsep Dasar Perancangan Sistem

2.2.1. Unified Modeling Language

Unified Modeling Language (UML) adalah keluarga notasi grafis yang didukung oleh meta-model tunggal, yang membantu pendeskripsian dan desain sistem perangkat lunak, khususnya sistem yang dibangun menggunakan pemrograman berorientasi objek[14].

UML dikeluarkan oleh OMG (*Object Management Group, Inc)* yaitu organisasi internasional yang dibentuk pada 1989, terdiri dari perusahaan sistem informasi, pengembang perangkat lunak dan pengguna sistem komputer.

UML merupakan kesatuan dari Bahasa permodelan yang dikembangkan booch, *Object Modeling Technique* (OMT) dan *Object Orented Software Engineering* (OOSE) Metode ini menjadikan proses analisis dan design ke dalam empat tahapan iterative, yaitu identifikasi kelas-kelas, dan objek-objek, identifikasi semantic dari hubungan objek dan kelas tersebut, perincian *interface* dan implementasi. Keunggulan metode Booch adalah pada detail dan kayanya dengan notasi dan elemen, permodelan OMT yang dikembangkan oleh Rumbaugh didasarkan pada analisis terstruktur dan permodelan *entity-relationship*. Tahapan utama dalam metodologi ini adalah analisis, desain sistem, desain objek dan implementasi. Keunggulan metode ini adalah dalam notasi yang mendukung konsep OO.

Metode OOSE dari Jacobson lebih memberi menekankan pada *use case*. OOSE memiliki tiga tahapan yaitu membuat model *requirement* dan analisis, desain dan implementasi, dan model pengujian. Keunggulan metode ini adalah mudah dipelajari karena memiliki notasi yang sederhana namun mencakup seluruh tahapan dalam rekayasa perangkat lunak. Dengan UML, metode Booch, OMT dan OOSE digabungkan dengan membuang elemen-elemen yang tidak praktis ditambah dengan elemen-elemen dari metode lain yang lebih efektif dan elemen-elemen baru yang belum ada pada metode terdahulu sehingga UML lebih ekspresif dan seragam daripada metode lainnya.

Sebagai sebuah notasi grafis yang relatif sudah dibakukan (*open standard*) dan dikontrol oleh OMG (*Object Management Group*) mungkin lebih dikenal sebagai badan yang berhasil membakukan CORBA (*Common Object Request Broker Architecture*), UML menawarkan banyak keistimewaan. UML tidak hanya dominan dalam notasi di lingkuangan OO tetapi juga popular di luar lingkungan OO. Paling tidak ada tiga karakter penting yang melekat di UML yaitu sketsa, cetak biru dan Bahasa pemrograman. Sebagai sebuah sketsa, UML bisa berfungsi

sebagai jembatan dalam mengkomunikasikan beberapa aspek dari sistem. Dengan demikian

semua anggota tim akan mempunyai gambaran yang sama tentang suatu sistem. UML bisa juga berfungsi sebagai sebuah cetak biru karena sangat lengkap dan detail. Dengan cetak biru ini maka akan bisa diketahui informasi detail tentang *coding* program (*forward engineering*) atau bahkan membaca program dan menginterpretasikannya kembali ke dalam diagram (*reverse engineering*). *Reverse engineering* sangat berguna pada situasi di mana *code* program yang tidak terdokumentasi asli hilang atau bahkan belum dibuat sama sekali. Sebagai bahasa pemrograman, UML dapat menerjemahkan diagram yang ada di UML menjadi *code* program yang siap untuk di jalankan.

Struktur sebuah sistem dideskripsikan dalam 5 *view* di mana salah satu di antaranya *scenario*, *scenario* ini memegang peran khusus untuk mengintegrasikan *content* ke *view* yang lain. Kelima *view* tersebut berhubungan dengan diagram yang dideskripsikan di UML. Setiap *view* berhubungan dengan perspektif tertentu di mana sistem akan diuji. *View* yang berbeda akan menekankan pada aspek yang berbeda dari siste yang mewakili ketertarikan sekelompok *stakeholder* tertentu. Penjelasan lengkap tentang sistem bisa dibentuk dengan menggabungkan informasi-informasi yang ada pada kelima *view* tersebut sebagai berikut :

1. *Scenario*, menggambarkan interaksi di antara objek dan di antara proses. *Scenario* ini digunakan untuk diidentifikasi elemen arsitektur, ilustrasi dan validasi desain arsitektur serta sebagai titik awal untuk pengujian purwa rupa arsitektur. *Scenario* ini bisa juga disebut dengan *use case view*. *Use case view* ini mendefinisikan kebutuhan sistem karena mengandung semua *view* yang lain yang mendeskripsikan aspek-aspek tertentu dari rancangan sistem. Itulah sebabnya *use case view* menajdi pusat peran dan sering dikatakan yang mengatur proses pengembangan perangkat lunak.
2. *Development View*, menjelaskan sebuah sistem dari perspektif programer dan terkonsentrasikan ke manajemen perangkat lunak. *View* ini dikenal juga sebagai *implementation view*. Diagram UML yang termasuk dalam

development *view* di antaranya adalah *component diagram* dan *package diagram*.

3. *Logical View*, terkait dengan fungsionalitas sistem yang dipersiapkan untuk pengguna akhir. *Logical view* mendeskripsikan struktur logika yang mendukung fungsi-fungsi yang dibutuhkan di *use case*. *Design view* ini berisi *object diagram*, *class diagram*, *state machine diagram* dan *composite structure diagram*.
4. *Physical View*, menggambarkan sistem dari perspektif *sistem engineer*. Fokus dari *physical view* adalah *topologi* sistem perangkat lunak. *View* ini dikenal juga sebagai deployment view. Yang termasuk dalam *physical* view ini adalah *deployment diagram* dan *timing diagram*.
5. *Process View*, berhubungan erat dengan aspek dinamis dari sistem, proses yang terjadi di sistem dan bagaimana komunikasi yang terjadi di sistem serta tingkah laku sistem saat dijalankan. *Process view* menjelaskan apa itu *concurrency*, *distribusi integrasi*, *kinerja* dan lain-lain. Yang termasuk dalam *process view* adalah *activiy diagram*, *communication diagram*, *sequence diagram* dan *interaction overview diagram*.

Meskipun UML sudah cukup banyak menyediakan diagram yang bisa membantu mendefinisikan sebuah aplikasi, tidak berarti bahwa semua diagram tersebut akan bisa menjawab persoalan yang ada. Dalam banyak kasus, diagram lain selain UML sangat banyak membantu. Pada penelitian ini konsep UML digunakan untuk menggambarkan bagaimana sistem bekerja, hubungan antar class, memberi gambaran kepada user sistem yang akan di gunakan dan memberikan penjelasan detail pemanggilan fungsi-fungsi atau *method* dalam *class*.

2.2.2. *Use Case Diagram*

Use Case adalah suatu pola urutan langkah-langkah yang menggambarkan interaksi antara sistem dan *actor* yang berhubungan dalam domain aplikasi[14]. *Use case* bekerja dengan cara mendeskripsikan tipikal interaksi antara user (pengguna) sebuah sistem dengan sistemnya sendiri melalui sebuah cerita

bagaimana sebuah sistem dipakai. Urutan langkah-langkah yang menerangkan antara pengguna dan sistem disebut *scenario*. Setiap *scenario* mendeskripsikan urutan kejadian[19]. Setiap urutan diinisialisasi oleh orang, sistem yang lain, perangkat keras atau urutan waktu. Dengan demikian secara singkat bisa dikatakan *use case* adalah satu rangkaian *scenario* yang digabungkan Bersama-sama oleh tujuan umum pengguna. Diagram use case menunjukkan 3 aspek dari sistem yaitu : *actor*, *use case* dan *sistem* atau *sub sistem boundary*. *Actor* mewakili peran orang, sistem yang lain atau alat ketika berkomunikasi dengan use case.

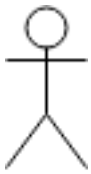
2.2.2.1. Use Case

Use case adalah abstraksi dari interaksi antara sistem dengan *actor*. Oleh karena itu sangat penting untuk memilih abstraksi yang cocok. *Use case* dibuat berdasarkan keperluan *actor*. *Use case* harus merupakan 'apa' yang dikerjakan perangkat lunak aplikasi, bukan 'bagaimana' perangkat lunak aplikasi mengerjakannya. Setiap *user case* harus diberi nama yang menyatakan apa hal yang dicapai dari hasil interaksinya dengan *actor*. Nama *use case* boleh terdiri dari beberapa kata dan tidak boleh ada dua *use case* yang memiliki nama yang sama.

Gambar 2.2 Simbol Use Case

2.2.2.2. Actor

Actors adalah *abstraction* dari orang dan sistem yang lain yang mengaktifkan fungsi dari target sistem. Orang atau sistem bisa muncul dalam beberapa peran. Bahwa actor berinteraksi dengan *use case*, tetapi tidak memiliki kontrol atas *use case*.



Gambar 2.3 Simbol Actor

2.2.2.3. Relationship

Relationship adalah hubungan antara *use cases* dengan *actors*. *Relationship* dalam *use case* diagram meliputi:

1. Asosiasi antara *actor* dan *use case*. Hubungan antara *actor* dan *use case* yang terjadi karena adanya interaksi antara kedua belah pihak. Asosiasi tipe ini menggunakan garis lurus dari actor menuju *use case* baik dengan menggunakan mata panah terbuka ataupun tidak.
2. Asosiasi antara 2 *use case* Hubungan antara *use case* yang satu dan *use case* lainnya yang terjadi karena adanya interaksi antara kedua belah pihak. Asosiasi tipe ini menggunakan garis putus-putus/garis lurus dengan mata panah terbuka di ujungnya.
3. Generalisasi antara 2 *actor* Hubungan *inheritance* (pewarisan) yang melibatkan *actor* yang satu (*the child*) dengan *actor* lainnya (*the parent*). Generalisasi tipe ini menggunakan garis lurus dengan mata panah tertutup di ujungnya.
4. Generalisasi antara 2 *use case*. Hubungan *inheritance* (pewarisan) yang melibatkan *use case* yang satu (*the child*) dengan use case lainnya(*the parent*). Generalisasi tipe ini menggunakan garis lurus dengan mata panah tertutup di ujungnya.

2.2.3. *Activity Diagram*

Activity diagram adalah teknik untuk menggambarkan logika prosedural, proses bisnis, dan jalur kerja. Dalam beberapa hal, *activity diagram* memainkan

peran mirip diagram alir, tetapi perbedaan prinsip antara notasi diagram alir adalah *activity diagram* mendukung *behavior* paralel. *Node* pada sebuah *activity diagram* disebut sebagai *action*, sehingga diagram tersebut menampilkan sebuah *activity* yang tersusun dari *action*[14].

Tujuan dari *activity diagram* adalah untuk menangkap tingkah laku dinamis dari sistem dengan cara menunjukkan aliran pesan dari satu aktivitas ke aktivitas lainya. Secara umum *activity diagram* digunakan untuk menggambarkan diagram alir yang terdiri dari banyak aktivitas dalam sistem dengan beberapa fungsi tambahan seperti percabangan, aliran pararel, *swimlane*, dan sebagainya. Sebelum menggambarkan sebuah *activity diagram,* perlu adanya pemahaman yang jelas tentang elemen yang akan digunakan di *activity diagram*. Elemen utama dalam *activity diagram* adalah aktivitas itu sendiri. Aktivitas adalah fungsi yang dilakukan oleh sistem Setelah aktivitas teridentifikasi, selanjutnya yang perlu diketahui adalah bagaimana semua elemen tersebut berasosiasi dengan constraint dan kondisi. Langkah selanjutnya perlu penjabaran tata letak dari keseluruhan aliran agar bisa di transformasikan ke activity *diagram*.

Berikut ini merupakan komponen dalam *activity diagram*, yaitu:

1. *Activity node Activity node* menggambarkan bentuk notasi dari beberapa proses yang beroperasi dalam kontrol dan nilai data.
2. *Activity edge Activity edge* menggambarkan bentuk *edge* yang menghubungkan aliran aksi secara langsung ,dimana menghubungkan *input* dan *output* dari aksi tersebut.
3. *Initial state* Bentuk lingkaran berisi penuh melambangkan awal dari suatu proses.
4. *Decision* Bentuk wajib dengan suatu *flow* yang masuk beserta dua atau lebih *activity node* yang keluar. *Activity node* yang keluar ditandai untuk mengindikasikan beberapa kondisi.
5. *Fork* Satu bar hitam dengan satu *activity node* yang masuk beserta dua atau lebih *activity node* yang keluar.

6. *Join* Satu bar hitam dengan dua atau lebih *activity node* yang masuk beserta satu *activity node* yang keluar, tercatat pada akhir dari proses secara bersamaan. Semua *actions* yang menuju *join* harus lengkap sebelum proses dapat berlanjut.
7. *Final state* Bentuk lingkaran berisi penuh yang berada di dalam lingkaran kosong, menunjukkan akhir dari suatu proses.

2.2.4. Class Diagram

Class diagram merupakan himpunan dari objek-objek yang sejenis. Sebuah objek memiliki keadaan sesaat (*state*) dan perilaku (*behavior*). *State* sebuah objek adalah kondisi objek tersebut yang dinyatakan dalam *attribute/properties*. Sedangkan perilaku suatu objek mendefinisikan bagaimana sebuah objek bertindak/beraksi dan memberikan reaksi[14].

Class diagram mempunyai 3 relasi dalam penggunaannya, yaitu:

1. *Assosiation*

   Assosiation adalah sebuah hubungan yang menunjukkan adanya interaksi antar *class*. Hubungan ini dapat ditunjukkan dengan garis dengan mata panah terbuka di ujungnya yang mengindikasikan adanya aliran pesan dalam satu arah.
2. *Generalization*

   Generalization adalah sebuah hubungan antar *class* yang bersifat dari khusus ke umum.
3. *Constraint*

   Constraint adalah sebuah hubungan yang digunakan dalam sistem untuk memberi batasan pada sistem sehingga didapat aspek yang tidak fungsional.

2.2.5. Sequence Diagram

Sequence diagram adalah grafik dua dimensi dimana objek ditunjukkan dalam dimensi horizontal, sedangkan *lifeline* ditunjukkan dalam dimensi vertikal[14].

Diagram ini menunjukan sejumlah contoh obyek dan *message* (pesan) yang diletakan diantara obyek-obyek ini di dalam use case. Komponen utama *sequence diagram* terdiri atas objek yang di tulis dengan kotak segi empat bernama. *Message* diwakili oleh garis dengan tanda panah dan waktu yang ditunjukkan dengan *progress vertical*. Objek diletakan di detak bagian atas diagram dengan urutan dari kiri ke kanan. Mereka diatur dalam urutan guna menyederhanakan diagram, istilah objek dikenal juga dengan *participant*, setiap *participant* terhubung dengan garis titik-titik yang disebut *lifeline*. Sepanjang *lifeline* ada kotak yang disebut *activation*, *activation* mewakili sebuah eksekusi operasi dari *participant*. Panjang kotak ini berbanding lurus dengan durasi *activation*. Sebuah *message* bisa jadi simpel, *synchronous* atau *asynchronous*. *Message* yang simpel adalah sebuah perpindahan (*transfer*) *control* dari satu *participant* ke *participant* yang lainnya. Jika sebuah *participant* mengirimkan sebuah message *synchronous*, maka jawaban atas *message* tersebut akan ditunggu sebelum diproses dengan urusannya. Namun jika *message asynchronous* yang dikirimkan, maka jawaban atas *message* tersebut tidak perlu ditunggu. *Time* adalah diagram yang mewakili waktu pada arah *vertical*. Waktu dimulai dari atas ke bawah. *Message* yang lebih dekat dari atas akan dijadikan terlebih dahulu dibanding *message* yang lebih dekat ke bawah.

Berikut ini merupakan komponen dalam *sequence diagra* :

1. *Activations* Activations menjelaskan tentang eksekusi dari fungsi yang dimiliki oleh suatu objek.
2. *Actor* *Actor* menjelaskan tentang peran yang melakukan rangkaian aksi dalam suatu proses.
3. *Collaboration boundary* *Collaboration boundary* menjelaskan tentang tempat untuk lingkungan percobaan dan digunakan untuk memonitor objek.
4. *Parallel vertical lines* *Parallel vertical lines* menjelaskan tentang suatu garis proses yang menunjuk pada suatu *state*.

5. *Processes* *Processes* menjelaskan tentang tindakan/aksi yang dilakukan oleh aktor dalam suatu waktu.
6. *Window* *Window* menjelaskan tentang halaman yang sedang ditampilkan dalam suatu proses.
7. *Loop* *Loop* menjelaskan tentang model logika yang berpotensi untuk diulang beberapa kali.

2.3. Perangkat Lunak Pendukung

2.3.1. Postgresql

PostgreSQL adalah basis data *open-source* yang paling canggih. Dikembangkan di Universitas Berkeley dipimpin oleh Michael Stonebraker pada tahun 1986-1994. Perangkat lunak *open-source* berarti tidak memiliki perusahaan. Dengan sistem seperti ini, pengembang memiliki kemungkinan untuk menyumbangkan ide-ide untuk memecahkan masalah[15].

Gambar 2.4 Logo Postgresql

Meskipun PostgreSQL menawarkan banyak fungsi, operator, tipe data dan agregat, terkadang pengguna masih perlu membuat sendiri fungsi yang dibutuhkan[15]. PostgreSQL menyediakan mekanisme fungsi yang memperbolehkan pengembang untuk mengimplementasikan berbagai extension ke *backend*–nya[16]. Salah satunya dengan mengembangkan (*Extending*) PostgreSQL menggunakan Bahasa C[15].

Pemilihan PostgreSQL karena kinerjanya yang baik dibandingkan basis data *open-source* lainya. Untuk proses *insert* 100.000 data dengan jenis yang sama

PostgreSQL unggul 7,6 Kali lebih cepat dibandingkan MySQL. Sedangkan untuk proses *delete* hampir 2 kali lebih cepat[4].

2.3.2. GO

Code less, compile quicker, execute faster => have more fun!. Kalimat yang menyatakan secara keseluruhan mengenai bahasa GO.

Beberapa pegawai Google merasakan frustasi saat mengembangkan sebuah perangkat lunak menggunakan C++. Membutuhkan waktu yang lama untuk *compile* dan bahasanya sudah "kuno". Banyak perangkat dan ide sudah berkembang dalam beberapa dekade terakhir namun tidak memiliki kesemapatan untuk dipengaruhi menggunakan bahasa C++. Yang mereka butuh kan adalah bahasa yang dapat menyelesaikan masalah :

1. Perangkat lunak harus dapat dibangun dengan cepat.
2. Bahasa harus dapat berjalan dengan baik di atas banyak *platform*.
3. Bahasa harus dapat berjalan dengan baik di jaringan komputer.
4. Bahasa harus dapat lebih mudah digunakan.

Lahirlah GO dengan bahasa yang dinamis seperti Python atau Ruby, tetapi memiliki performa dan keamanan seperti C atau Java. Dibangun dengan mengembangkan dari bahasa pendahulunya C/Java/C#[17].

Gambar 2.5 Logo Go Language

GO adalah bahasa baru yang memperkenalkan banyak fitur yang menarik yang menjadikan aspek pemrograman menjadi lebih mudah. GO memanfaatkan

model memori sederhana untuk secara otomatis mengelola beberapa eksekusi, yang dapat jauh lebih mudah dari pada pendekatan manual yang dilakukan oleh C.

Karena desain awal GO simpel, sebenarnya GO tidak bisa menjadi satu level dengan C. Untuk waktu *compile*, GO dapat mengungguli C namun dalam performa sebuah perangkat lunak yang besar, C lebih baik. Tujuan GO adalah memiliki pendekatan performa dengan C dan membuat *software engineering* lebih mudah[18].

2.3.3. SQL

SQL (*Structured Query Language*) adalah sebuah bahasa yang digunakan untuk mengakses data dalam basis data relasi. Bahasa ini secara *de facto* merupakan bahasa standar yang digunakan dalam manajemen basis data relasi. Saat ini hampir semua server basis data yang ada mendukung bahasa ini untuk melakukan manajemen datanya[19].