BAB II

TINJAUN PUSTAKA

2.1 Tinjauan Perusahaan

Pada sub bab ini akan menjelaskan mengenai profil perusahaan yang menjelaskan tentang sejarah berdirinnya perusahaan, Visi maupun misi perusahaan serta sturktur organisasi perusahaan.

2.1.1 Sejarah Perusahaan [1]

Kehadiran kereta api di Indonesia ditandai dengan pencangkulan pertama pembangunan jalan KA didesa Kemijen Jumat tanggal 17 Juni 1864 oleh Gubernur Jenderal Hindia Belanda, Mr. L.A.J Baron Sloet Van Den Beele. Pembangunan diprakarsai oleh Naamlooze Venootschap Nederlandsch Indische Spoorweg Maatschappij (NV. NISM) yang dipimpin oleh Ir. J.P De Bordes dari Kemijen menuju Desa Tanggung sepanjang 26 Km dengan lebar sepur 1435 mm. Ruas jalan ini dibuka untuk angkutan umum pada hari sabtu, 10 agustus 1867.

Keberhasilan swasta, NV. NISM membangun jalan KA antara Kemijen sampai Tanggung, yang kemudian pada tanggal 10 februari 1870 dapat menghubungkan kota Semarang dan Surakarta sepanjang 110 Km, akhirnya mendorong minat investor untuk membangun jalan KA didaerah lainnya.

Setelah kemerdekaan Indonesia diproklamasikan pada tanggal 17 Agustus 1945, karyawan KA yang tergabung dalam Angkatan Moeda Kereta Api (AMKA)

mengambil alih kekuasaan perkeretaapian dari pihak Jepang. Peristiwa bersejarah yang terjadi pada tanggal 28 September 1945, pembacaan pernyataan sikap oleh Ismangil dan sejumlah anggota AMKA lainnya, menegaskan bahwa mulai tanggal 28 September 1945 kekuasaan perkeretaapian berada ditangan bangsa Indonesia. Orang Jepang tidak diperkenankan lagi campur tangan dengan urusan perkeretaapian di Indonesia. Inilah yang melandasi ditetapkannya 28 September 1945 sebagai Hari Kereta Api di Indonesia, serta dibentuknya Djawatan Kereta Api Republik Indonesia (DKARI).

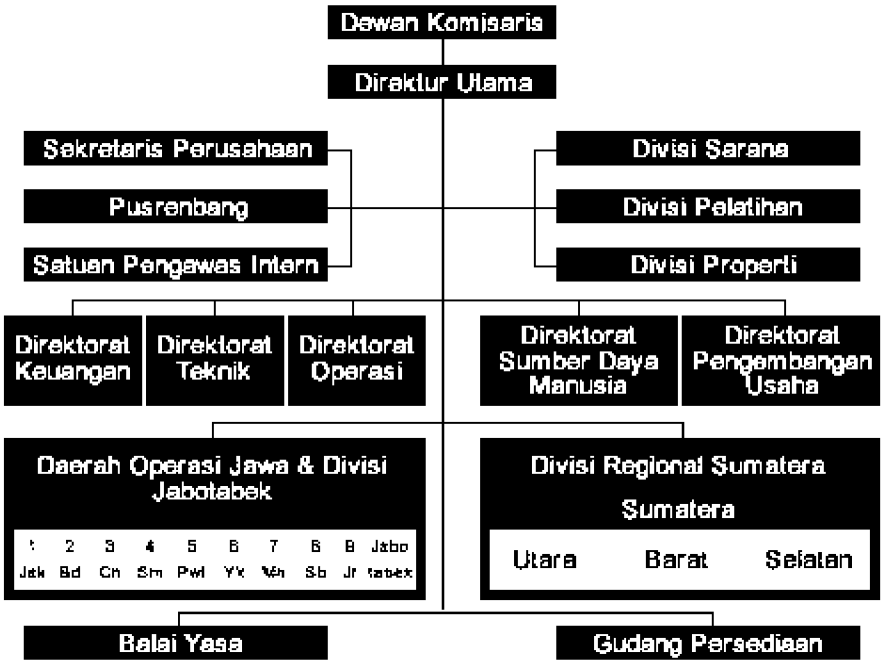
2.1.2 Visi dan Misi Perusahaan

Visi dari perusahaan PT.KAI adalah terwujudnya kereta api sebagai pilihan utama jasa transportasi dengan fokus keselamatan dan pelayanan yang terbaik kepada masyarakat sedangkan misi dari perusahaan yaitu menyelenggarakan jasa transportasi sesuai keinginan masyarakat dengan meningkatkan keselamatan dan pelayanan serta penyelenggaraan yang semakin efisien [1] .

2.1.3 Struktur Organisasi

Struktur organisasi merupakan susunan seluruh organisasi dari PT Kereta Api Indonesia, mulai dari yang tertinggi yaitu dewan komisaris sampai ke daerah operasi dan divisi regional. Organisasi tertinggi dalam struktur organisasi PT.KAI adalah dewan komisaris yang menjabat sebagai dewan tertinggi diatas direktur utama yang memiliki kemampuan dan keputusan menganai arah susunan organisasi PT.KAI

sesuai dengan keputusan mentri transportasi indonesia, semua susunan organisasi didalam PT.KAI mulai dari dewan komisaris sampai ke daerah operasi dan divisi regional memiliki fungsi dan perannnya masing-masing untuk menuju satu tujuan yang sama yaitu memiliki visi yang sama memberikan kenyamanan bertransportasi berkereta api kepada masyarakat. Untuk lebih jelasnya dapat dilihat pada gambar dibawah ini [1]:



Gambar 2.1 Struktur Organisasi

2.1.4 Tugas Dan Wewenang

Tugas dan wewenang dilakukan pada bagian pusrenbang sebagai pusat pengembangan IT dan implementasi yang berada dikantor pusat milik PT.KAI dijalan perintis kemerdekaan nomor 1 Bandung, sedangkan tugas yang ditiliti berada pada

daerah operasional jawa dan divisi JABOTABEK dengan studi kasus menjadwalkan KRL pada bagaian dipo Depok divisi dari JABODETABEK milik PT.KAI.

2.2 Landasan Teori

Pada subab landasan teori ini akan menjelaskan beberapa teori yang akan berhubungan dengan permasalahan yang akan dibahas sebagai dasar pemahaman dalam sebuah sistem serta metode yang dipakai untuk kegiatan pengembangan terhadap sistem.

2.2.1 Pengertian Sisitem Informasi

Terdapat beberapa istilah pengertian sistem informasi diantarannya pengertian sistem informasi menurut Henry C. Lucas, John F. Nash dan Martin B. Robert:

1. Menurut Henry C. Lucas

   "***Sistem Informasi adalah kegiatan dari suatu prosedur-prosedur yang diorganisasikan bilamana dieksekusi akan menyediakan informasi untuk mendukukung pengambilan keputusan dan pengendalian di dalam organisasi***".

2. Menurut John F. Nash dan Martin B. Robert

   "Sistem Informasi adalah suatu kombinasi dari orang-orang, fasilitas teknologi, media, prosedur-prosedur dan pengendalian ditujukan untuk mendapatkan jalur komunikasi penting, memproses tipe transaksi rutin tertentu, memberi sinyal kepada manajemen dan yang lainnya terhadap

kejadian-kejadian internal dan eksternal yang penting dan menyediakan suatu dasar untuk pengambilan keputusannya yang cerdik".

Berdasarkan pengertian sistem informasi yang telah disebutkan oleh Henry C. Lucas, John F. Nash dan Martin B. Robert tersebut sistem informasi memiliki suatu kelebihan diantarannya untuk mengambil sebuah keputusan berdasarkan ketentuan-ketentuan yang telah ada untuk mendukung sistem informasi tersebut.

2.2.2 Landasan Dasar Algoritma Genetik

Algoritma genetik ditemukan di Universitas Michigan, Amerika Serikat oleh John Holland (1975) melalui sebuah penelitian dan dipopulerkan oleh salah satu muridnya, David Goldberg. Algoritma genetik adalah algoritma yang berusaha menerapkan pemahaman mengenai *evolusi* alamiah pada tugas-tugas pemecahan masalah (*problem solving*). Pendekatan yang diambil oleh algoritma ini adalah dengan menggabungkan secara acak berbagai pilihan solusi terbaik didalam suatu kumpulan untuk mendapatkan generasi solusi terbaik berikutnya yaitu pada suatu kondisi yang memaksimalkan kecocokannya atau lazim disebut *fitness*. Generasi ini akan merepresentasikan perbaikan-perbaikan pada populasi awalnya, dengan melakukan proses ini secara berulang, algoritma ini diharapkan dapat mensimulasikan proses *evolusione*r. Pada akhirnya, akan didapatkan solusi-solusi yang paling tepat bagi permasalahan yang dihadapi. Untuk menggunakan algoritma genetik, solusi permasalahan direpresentasikan sebagai kromosom. Tiga aspek yang penting untuk penggunaan algoritma genetik:

1 Definisi *fitness function*

2. Definisi dan implementasi representasi genetik

3. Definisi dan implementasi operasi genetik

Jika ketiga aspek diatas telah didefinisikan, algoritma genetik genetik akan bekerja dengan baik. Tentu saja, algoritma genetik bukanlah solusi terbaik untuk memecahkan segala masalah. Sebagai contoh, metode tradisional telah diatur untuk untuk mencari penyelesaian dari fungsi *analitis convex* yang variabelnya sedikit.

Pada kasus-kasus ini, metode berbasis kalkulus lebih unggul dari algoritma genetik karena metode ini dengan cepat menemukan solusi minimum ketika algoritma genetik masih menganalisa bobot dari populasi awal. Untuk problem-problem ini pengguna harus mengakui fakta dari pengalaman ini dan memakai metode tradisional yang lebih cepat tersebut. Akan tetapi, banyak persoalan realistis yang berada diluar golongan ini. Selain itu, untuk persoalan yang tidak terlalu rumit, banyak cara yang lebih cepat dari algoritma genetik. Jumlah besar dari populasi solusi, yang merupakan keunggulan dari algoritma genetik, juga harus mengakui kekurangannya dalam dalam kecepatan pada sekumpulan komputer yang dipasang secara seri *fitness function* dari tiap solusi harus dievaluasi. Namun, bila tersedia komputer-komputer yang paralel, tiap prosesor dapat mengevaluasi fungsi yang terpisah pada saat yang bersamaan. Karena itulah, algoritma genetik sangat cocok untuk perhitungan yang parallel [6].

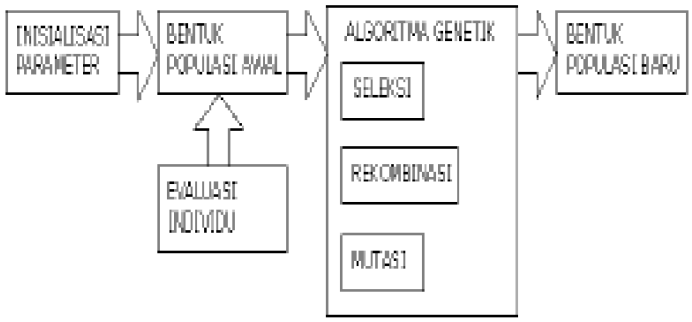
2.2.3 Penggunaan Algoritma Genetika

Algoritma genetik dimulai dengan sekumpulan set status yang dipilih secara random, yang disebut populasi. Algoritma ini mengkombinasikan dua populasi induk. Setiap status atau individual direpresentasikan sebagai sebuah string. Setiap individual dievaluasi dengan *fitness function*. Sebuah *fitness function* mengembalikan nilai tertinggi untuk individual yang terbaik. Individu akan diurutkan berdasarkan nilai atau disebut dengan *selection*.

Untuk setiap pasang induk, sebuah titik *crossover* akan dipilih secara acak dari posisi dalam string. Pada gambar titik *crossover* terletak pada indeks ketiga dalam pasangan pertama dan setelah indeks kelima pada pasangan kedua.

Pada mutasi, tiap lokasi menjadi sasaran mutasi acak, dengan *probabilitas independen* yang kecil. Sebuah digit dimutasikan pada anak pertama, ketiga, dan keempat. Algoritma genetik mengkombinasikan suatu kecenderungan menaik dengan pengeksplorasian acak diantara *thread* pencarian paralel. Keuntungan utamanya,bila ada, datang dari operasi *crossover*. Namun, secara matematis dapat tunjukkan bahwa bila posisi dari kode genetik dipermutasikan diawal dengan urutan acak, *crossover* tidak memberikan keunggulan. Secara intuisi, keuntungannya didapat dari kemampuan *crossover* untuk menggabungkan blok-blok huruf berukuran besar yang telah berevolusi secara independen untuk melakukan fungsi yang bermanfaat sehingga dapat menaikkan tingkat *granularity* dimana pencarian dilakukan.

Teori dari algoritma genetik menjelaskan cara kerjanya menggunakan ide dari suatu sekema, suatu sub-string dimana beberapa posisi tidak disebutkan. Dapat ditunjukkan bahwa, bila *fitness* rata-rata dari *schema* berada dibawah *mean* maka jumlah instansiasi dari sekema didalam populasi akan bertambah seiring bertambaahnya waktu. Jelas sekali bahwa efek ini tidak akan signifikan bila bit-bit yang bersebelahan sama sekali tidak berhubungan satu sama sekali, karena akan ada beberapa blok yang memberikan keuntungan yang konsisten. Algoritma genetik paling efektif dipakai bila skema-skema berkorespondensi menjadi komponen berati dari sebuah solusi. Sebagai contoh, bila *string* adalah representasi dari sebuah antena, maka schema merepresentasikan komponen-komponen dari antena, seperti *reflector* dan *deflector*. Sebuah komponen yang baik cenderung akan berkerja baik pada rancangan yang berbeda. Ini menunjukkan bahwa penggunaan algoritma genetik yang benar memerlukan rekayasa yang baik.



Gambar 2.2 Siklus Dasar Algoritma Genetik

2.2.3.1 Skema Pengkodean

Misalkan kita ingin memecahkan masalah optimasi fungsi produksi Cobb-Dauglas yaitu y = β1Lβ2Kβ3 dengan sample yang ada untuk L dan K berapa nilai β1, β2, β3 dengan fungsi tujuan meminimumkan least square atau memaksimumkan fungsi likelihood. Dengkian pula untuk persoalan yang sama pada fungsi produksi CES. Persoalan tersebut dapat diselesaikan dengan AG, yaitu: ketiga parameter β1, β2, β3 dikodekan dalam kromosom. Masing- masing kromosom berisi sejumlah gen, yang mengkodekan informasi yang disimpan di dalam kromosom. Misalkan untuk memudahkan digunakan binary encoding dengan panjang kromosom 12 gen (12 bits), masing-masing parameter β1, β2, β3 dikodekan dengan 4 gen, sehingga dapat iilustrasikan skema pengkodean pada Gambar 2.3 bawah ini:

Parameter	β_1				β_2				β_3			
Binary number	1	0	1	1	1	1	1	0	1	0	1	0
	g_1			g_4	g_5			g_8	g_9			g_{12}
Decimal number	11				14				3			

Gambar 2.3 Skema Binary Encoding

Bilamana nilai parameter yang akan kita cari mempunyai konstraint. yaitu a < β < b maka berdasarkan binary encoding, nilai parameter dapat dieroleh dengan formula :β

= a + βdec ∗ ((b−a)

2n−1)) dan misalkan n adalah

banyaknya gen (bits) yaitu 4 untuk setiap parameter dan kontraint $0 < \beta < 1$, Setetelah skema pengkodean ditentukan, AG diinisialisasi untuk sebuahpopulasi dengan N kromosom. Gen-gen yang mengisi masing- masing kromosom dibangkitkan secara random. Masing- masing kromosom akan dikodekan menjadi individu dengan nilai fitness tertentu. Kemudian sebuah populasi baru akan dibentuk dengan menggunakan mekanisme seleksi alamiah, yaitu memilih individu- individu secara proporsional terhadap nilai fitnessnya, dan genetika alamiah, yakni pindah silang (crossover) dan mutasi.Pada algoritma getika yang akan digunakan adalah dengan skema pergantian populasi yang disebut generational replacement, artinya, N kromosom dari suatu generasi digantikan sekaligus oleh N kromosom baru hasil pindah silang dan mutasi.

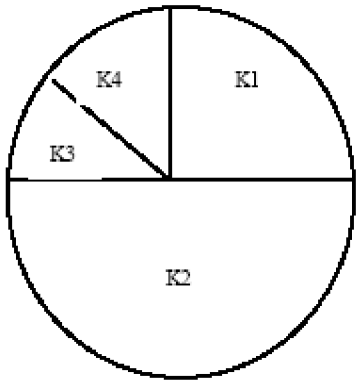
2.2.3.2 Nilai Fitness

Suatu individu dievaluasi berdasarkan suatu fungsi tertentu sebagai ukuran performansinya. Di dalam evolusi alam, individu yang bernilai fitnes tinggi yang akan bertahan hidup. Sedangkan individu yang bernilai fitness rendah akan mati. Pada masalah optimasi, solusi yang akan dicari adalah memaksimumkan sebuah fungsi likelihood dan meminimumkan least square baik untuk fungsi produksi Cobb-Dauglas maupun fungsi produksi CES.

2.2.3.3 Seleksi Orang Tua

Pemilihan dua buah kromosom sebagai orang tua, yang akan dipindahsilangkan, biasanya dilakukan secara proporsional sesuai dengan dengan nilai fitness-nya. Suatu meetoda seleksi yang umumnya digunakan adalah roulette wheel (roda raoulette). Sesuai dengan namanya, metoda ini menirukan permainan roulette wheel di mana masing-masing kromosom menempati potongan lingkaran pada roda raulette secara proporsional sesuai dengan nilai fitnessnya. Kromosom yang memiliki nilai fitness lebih besar menempati potongan lingkaran yang lebih besar dibandingkan dengan kromosom bernilai fitness rendah.

Komosom	Nilai Fitness
K1	1
K2	2
K3	0,5
K4	0,5
Jumlah	4



Gambar 2.4 Contoh penggunaan metoda roulette wheel selection.

Metoda raulette-wheel selection sangat mudah diimplementasikan dalam pemprograman. Pertama, dibuat interval nilai kumulatif dari nilai fitness masing-masing kromosom. Sebuah kromosom akan terpilih jika bilangan random yang dibangkitkan berada dalam interval kumulatifnya. Pada Gambar 2.4 di atas, K1 menempati interval kumulatif [0;0,25], K2 berada dalam interval (0,25;0,74], K3

dalam interval (0,75;0,875] dan K4 berada dalam interval (0,875;1]. Misalkan, jika bilangan random yang dibangkitkan adalah 0,6 maka kromosom K2 terpilih sebagai orang tua. Tetapi jika bilangan random yang dibangkitkan adalah 0,9 maka kromosom K4 yang terpilih.

2.2.3.4 Pindah Silang (Cross-over)

Salah satu komponen yang paling penting dalam algoritma genetik adalah crossover atau pindah silang. Sebuah kromosom yang mengarah pada solusi yang baik dapat diperoleh dari proses memindah-silangkan dua buah kromosom.

	β_1				β_2				β_3			
Orang tua 1	**0**	**0**	1	1	1	1	1	1	1	1	1	1
Orang tua 2	1	1	0	0	0	0	0	0	0	0	0	0
	g_1			g_4	g_5			g_8	g_9			g_{12}
Anak 1	**0**	**0**	0	0	0	0	0	0	0	0	0	0
Anak 2	1	1	1	1	1	1	1	1	1	1	1	1

Gambar 2.5 Contoh Proses Pindah Silang

Pindah silang juga dapat berakibat buruk jika ukuran populasinya sangat kecil. Dalam suatu populasi yang sangat kecil, suatu kromosom dengan gen-gen yang mengarah ke solusi akan sangat cepat menyebar ke kromosomkromosom lainnya. Untuk mengatasi masalah ini digunakan suatu aturan bahwa pindah silang hanya bisa dilakukan dengan suatu probabilitas tertentu, artinya pindah silang bisa dilakukan hanya jika suatu bilangan random yang dibangkitkan kurang dari probabilitas yang ditentukan

tersebut. Pada umumnya probabilita tersebut diset mendekati 1. Pindah silang yang paling sederhana adalah pindah silang satu titik potong (one-point crossover). Suatu titik potong dipilih secara random, kemudian bagian pertama dari orang tua 1 digabungkan dengan bagian kedua dari orang tua 2 (terlihat pada gambar 2.5). Crossover adalah operator Algoritma Genetika yang utama karena beroperasi pada dua kromosom pada suatu waktu dan membentuk offspring dengan mengkombinasikan dua bentuk kromosom. Cara sederhana untuk memperoleh crossover adalah dengan memilih suatu titik yang dipisahkan secara random dan kemudian membentuk offspring dengan cara mengkombinasikan segmen dari satu induk ke sebelah kiri dari titik yang dipisahkan dengan segmen dari induk yang lain ke sebelah kanan dari titik yang dipisahkan. Metode ini akan berjalan normal dengan representasi bit string. Performa dari Algoritma Genetika bergantung pada performa dari operator crossover yang digunakan. Crossover rate merupakan rasio antara jumlah offspring yang dihasilkan pada setiap generasi terhadap luas populasinya. Semakin tinggi crossover rate akan memungkinkan eksplorasi ruang solusi yang lebih luas dan mereduksi kemungkinan jatuh pada kondisi optimum yang salah. Namun memberikan rate yang memberikan konsekuensi makin lamanya waktu perhitungan yang diperlukan sebagai akibat eksplorasi pada luas populasi yang ada.

2.2.3.5 Mutasi

Mutasi dapat dilakukan dari semua gen yang ada dengan probabilitas mutasi tertentu. Jika bilangan random yang dibangkitkan kurang dari probabilitas mutasi

yang ditentukan maka ubah gen tersebut menjadi nilai kebalikan yang dalam hal ini, binary encoding, 0 diubah 1, dan 1 diubah 0. Bila mana probabilitas mutasi adalah (1 12) maka sebanyak 1 gen akan dimutasi dari kromosom yang terdiri dari 12 gen (bits). Pada algoritma genetika yang sederhana, nilai probabilitas mutasi adalah tetap selama evolusi. Gambar 2.6 menunjukan proses mutasi yang terjadi pada gen5.

	β_1				β_2				β_3			
Kromosom asal	0	0	0	1	1	1	1	1	1	1	1	1
	g_1			g_4	g_5			g_8	g_9			g_{12}
Hasil mutasi	0	0	0	1	0	1	1	1	1	1	1	1

Gambar 2.6 Contoh Proses Mutasi

Mutasi dapat dikatakan sebagai operasi pendukung yang menghasilkan perubahan secara acak dan seketika pada berbagai jenis kromosom. Cara mudah untuk mendapatkan mutasi dengan mengubah satu atau lebih genes. Pada Algoritma Genetika, mutasi memainkan peran penting, yaitu pertama, menggantikan genes yang hilang dari populasi selama proses seleksi, sehingga dapat diujikan pada suatu kondisi yang baru. Kedua, menyediakan genes yang tidak ditampilkan pada populasi awal. utation rate menyatakan presentase dari total jumlah genes dalam opulasi. Mutation rate ini melakukan kontrol dimana genes baru dalam populasi dapat diuji seleksi. Jika rate terlalu kecil akan banyak genes yangsebenarnya bermanfaat tetapi tidak pernah diuji seleksi. Namun jika rate terlalu tinggi akan terjadi random pertubation, yang berakibat offspring mulai kehilangan kemiripan dengan induknya dan Algoritma Genetika akan kehilangan kemampuan untuk melihat urutan langkah observasinya.

2.2.3.6 Elitisme

Proses seleksi dilakukan secara random sehingga tidak ada jaminan bahwa suatu indvidu yang bernilai fitness tertinggi akan selalu terpilih. Walaupun individu bernilai fitness tertinggi terpilih, mungkin saja individu tersebut akan rusak (nilai fitnessnya menurun) karena proses pindah silang. Oleh karena itu, untuk menjaga agar individu bernilai fitness tertinggi tersebut tidak hilang selama evolusi, maka perlu dibuat satu atau beberapa kopinya. Prosedure ini dikenal sebagai elitisme.

2.2.4 Entity Relationship Diagram (ERD)

Entity Relationship Diagram (ERD) adalah model konseptual yang mendeskripsikan hubungan antara penympanan (dalam DFD). ERD digunakan untuk memodelkan struktur data dan hubungan antar data. Dengan ERD model dapat diuji dengan mengabaikan proses yang dilakukan.

ERD pertama kali dideskripsikan oleh Peter Chen yang dibuat sebagai bagian dari perangkat lunak *CASE*.

Didalam atribut ada yang berfungsi sebagai *key*. *Key* adalah suatu atribut yang sifatnya unik. *Key* dapat dibangun dari satu atribut atau gabungan dari beberapa atribut. *Key* terbagi menjadi beberapa jenis, diantarnya [3] :

1. *Super key*, seluruh atribut dalam suatu entitas dijadikan *key*.
2. *Primary key*, 1 atribut dalam suatu entitas dijadikan *key*.
3. *Candidat key*, beberapa atribut dijadikan *key*.
4. *Foreign key*, suatu atribut yang berasal dari tabel yang berelasi.

Di dalam ERD, relasi, dapat terdiri dari sejumlah entitas yang disebut dengan derajat relasi. Derajat relasi maksimum disebut dengan kardinalitas sedangkan derajat minimum disebut dengan *modalitas*. Jadi, kardinalitas relasi menunjukkan jumlah maksimum entitas yang dapat berelasi dengan entitas pada himpunan entitas lain. Kardinalitas relasi yang terjadi diantara dua himpunan entitas (misalnya A dan B) dapat berupa:

1. Satu ke satu (*one to one* / 1-1)

Setiap entitas pada himpunan entitas A dapat berelasi dengan paling banyak satu entitas pada himpunan entitas B, demikian juga sebaliknya.

2. Satu ke banyak (*one to many* / 1-N)

Setiap entitas pada himpunan entitas A dapat berelasi dengan banyak entitas pada himpunan B, tetapi tidak sebaliknya.

3. Banyak ke banyak (*many to many* / N-N)

Setiap entitas pada himpunan entitas A dapat berelasi dengan banyak entitas pada himpunan entitas B, demikian juga sebaliknya. Relasi akan dipetakan menjadi tabel beserta atributnya.

2.2.5 Data Flow Diagram (DFD)

DFD adalah suatu model logika data atau proses yang dibuat untuk menggambarkan dari mana asal data dan kemana tujuan data keluar dari sistem, dimana data disimpan, proses apa yang menghasilkan data dan interaksi antara data yang tersimpan dan proses yang dikenakan pada data tersebut.

DFD sering digukan untuk menggambarkan suatu sistem yang telah ada atau sistem baru yang akan dikembangkan secara logika tanpa mempertimbangkan lingkungan fisik dimana data tersebut mengalir atau simana data tersebut akan disimpan.

DFD merupakan alat yang digunakan pada metodologi pengembangan sistem yang terstruktur. Kelebihan utama DFD yaitu:

1 Kebebasan dalam menjalankan implementasi teknik sistem.
2 Pemahaman lebih jauh mengenai keterkaitan satu sama lain dalam sistem dan subsistem.
3 Mengkomunikasikan pengetahuan sistem yang ada dengan pengguna melalui diagram aliran data.
4 Menganalisa system yang diajukan untuk mementukan apakah data-data dan proses yang diperlukan sudah ditetapkan.
5 Dapat digunakan sebagai latihan yang bermanfaat bagi penganalisis, sehingga bisa memahami dengan lebih baik keterkaitan satu sama lain dalam sistem dan subsistem.
6 Membedakan sistem dari lingkungannya dengan menempatkan batas-batasnya.
7 Dapat digunakan sebagai suatu perangkat untuk berinteraksi dengan pengguna.
8 Memungkinkan penganalisis menggambarkan setiap komponen yang digunakan dalam diagram.

DFD terdiri dari *context* diagram dan diagram rinci (*DFD Levelled*). *Context diagram* berfungsi memetakan model lingkungan (menggambarkan hubungan antara entitas luar, masukan dan keluaran sistem), yang direpresentasikan dengan lingkaran tunggal yang mewakili keseluruhan sistem. *DFD levelled* menggambarkan sistem sebagai jaringan kerja antara fungsi yang berhubungan satu sama lain dengan aliran dan penyimpanan data, model ini hanya memodelkan system dari sudut pandang fungsi.

Dalam *DFD levelled* akan terjadi penurunan level dimana dalam penurunan level yang lebih rendah harus mampu merepresentasikan proses tersebut ke dalam spesifikasi proses yang jelas. Setiap penurunan hanya dilakukan bila perlu. Aliran data yang masuk dan keluar pada suatu proses dilevel x harus berhubungan dengan aliran data yang masuk dan keluar pada level x+1 yang mendefinisikan proses pada level x tersebut.

Dalam penggambaran DFD, ada beberapa peraturan yang harus diperhatikan sehingga dalam penggambarannya tidak terjadi kesalahan, aturan tersebut yaitu:

1 Antar entitas tidak diijinkan terjadi hubungan atau relasi.
2 Tidak boleh ada aliran data antara entitas eksternal dengan data store.
3 Untuk alasan kerapian, entitas eksternal atau data store boleh digambar beberapa kali dengan tanda khusus, misalnya diberi nomor atau garis miring.
4 Data aliran data boleh mengalirkan beberapa paket data.
5 Bentuk anak panah aliran data boleh bervariasi.
6 Semua objek harus mempunyai nama.

7 Aliran data selalu diawali atau diakhiri dengan proses.

8 Semua aliran data harus mempunya tanda arah.

9 Jumlah proses tidak lebih dari sembilan proses dalam sistem, jika melebihi maka sebaiknya dikelompokan menjadi beberapa proses yang bekerja bersama-sama didalam suatu subsistem.

2.2.6 Flow Chart

Flow Chart adalah suatu bagan yang menggambarkan arus logika dari data yang akan diproses dalam suatu program dari awal sampai akhir. *Flow chart* merupakan alat yang berguna bagi programmer untuk mempersiapkan program yang rumit. *Flow chart* terdiri dari simbol-simbol yang mewakili fungsi-fungsi langkah program dari garis alir (*flow lines*) yang menunjukan urutan dari simbol-simbol yang akan dikerjakan.

2.2.7 Pengujian

Pengujian adalah elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain, dan pengkodean. Pentingnya pengujian perangkat lunak dan implikasinya yang mengacu pada kualitas perangkat lunak tidak dapat terlalu ditekan karena melibatkan sederetan aktivitas produksi dimana peluang terjadinya kesalahan manusia sangat besar dan arena ketidakmampuan manusia untuk melakukan dan berkomunikasi dengan sempurna

maka pengembangan perangkat lunak diiringi dengan aktivitas jaminan kualitas. Pendekatan Desain *Test Case* terbagi menjadi :

2.2.7.1 Pengujian White Box

Pengujian *White Box* merupakan pendekatan terhadap pengujian yang diturunkan dari pengetahuan struktur dan implementasi perangkat lunak.Pengujian *White Box* biasanya diterapkan untuk unit program yang relatif kecil seperti subrutin atau operasi yang terkait dengan suatu objek. Penguji dapat menganalisis kode dan menggunakan pengetahuan mengenai struktur komponen untuk menurunkan data uji [2]. Analisis kode dapat digunakan untuk menemukan berapa kasus uji yang dibutuhkan untuk menjamin bahwa semua statement pada program atau komponen dieksekusi paling tidak satu kali pada proses pengujian. Pengetahuan mengenai algoritma yang digunakan untuk implementasi beberapa fungsi dapat dipakai untuk mengidentifikasi partisi ekuivalensi lebih lanjut.

2.2.7.2 Pengujian Black Box

Pengujian *Black Box* merupakan pendekatan pengujian yang ujinya diturunkan dari spesifikasi program atau komponen. Sistem merupakan kotak hitam yang perilakunya hanya dapat ditentukan dengan mempelajari input dan output yang berkaitan. Nama lain untuk cara ini adalah pengujian fungsional karena penguji hanya berkepentingan dengna fungsionalitas dan bukan implementasi perangkat

lunak.Pengujian *Black Box* dapat diterapkan pada sistem yang disusun sebagai sistem maupun sebagai objek [2]. Penguji memberikan input kepada komponen atau sistem dan meneliti output yang dihasilkan. Jika output bukan merupakan yang dikehendaki, berarati pengujian tersebut telah mendeteksi masalah dengan perangkat lunak tersebut.

2.2.8 Software Pendukung

Untuk membangun sebuah Sistem Informasi Penjadwalan KRL dibutuhkan perangkat lunak yang menunjang yaitu berupa program aplikasi *Borland C++ Builder 6* dengan *Database Management System (DBMS)* menggunakan *Interbase 7.1*.

2.2.8.1 Borland C++ Builder 6 [4]

C++ Builder adalah suatu alat pengembang aplikasi (*development tool*) berbasis *Microsoft Windows* yang menerapkan konsep visualisasi. Dengan adanya dukungan visualisasi ini *C++ Builder* menjadi mudah digunakan untuk membuat aplikasi-aplikasi secara cepat. Dengan *C++ Builder*, dapat melakukan *desain, testing, debugging,* maupun proses *deployment* aplikasi secara mudah.

Bahasa dasar yang digunakan dalam pemrograman *C++ Builder* adalah bahasa C++, yang telah terkenal dalam keampuhannya dalam pembuatan program. Memang harus diakui bahwa tingkat keteraturan dan kemudahan dari bahasa C++ relatif lebih rendah dibandingkan bahasa *Objek Pascal*, namun dengan mempelajari konsep-

konsep yang ada didalamnya, tentu C++ akan menjadi bahasa yang mudah dipahami dan digunakan.

Lingkungan yang terdapat pada C++ *Builder* menawarkan kemudahan dan kenyamanan dalam proses pengembangan aplikasi, sehingga mudah digunakan oleh siapapun, termasuk bagi para programmer pemula sekalipun. Namun bila digunakan oleh programmer yang telah berpengalaman, *C++ Builder* tentu akan dapat menghasilkan aplikasi yang berkemampuan tinggi.

Untuk memulai pemrograman dengan C++ *Builder* tentu harus mengenal dengan baik lingkungan yang ada didalamnya. Lingkungan semacam ini sering disebut dengan IDE *(Integrated Development Environtment)*.

2.2.8.2 Borland Interbase 7.1

Borland Interbase 7.1 adalah database transaksional yang ekonomis dan memiliki performa yang tinggi, dan banyak digunakan oleh jutaan pengguna diseluruh dunia. Dengan mengkombinasi instalasi yang mudah, *crash-recovery* secara *otomatis,* dan perawatan yang sangat minim, *Interbase* menjadi sangat sesuai untuk *embedding* dalam aplikasi terdistribusi. Dukungan terhadap *multiprosesor* dan arsitektur yang canggih menjadikannya sebuah pilihan yang tepat untuk aplikasi bisnis dengan kekuatan tinggi yang memiliki banyak pengguna yang terkoneksi. Kekuatannya, yaitu kemudahan penggunaan, dukungan terhadap *platform Windows, Linux*, dan *Solaris* (termasuk pengembangan dalam lingkungan seperti *Borland*

Delphi, *C++ Builder*, dan *Kylix*), membuat *Interbase* menjadi favorit bagi para pengembang.

Fitur-fitur dan keunggulan yang dimiliki oleh *interbase* adalah sebagai berikut:

a. Database terintegrasi
b. Cepat
c. Fleksibel
d. Dapat dipercaya
e. Kebebasan platform
f. Dukungan untuk standar industri

Adapula arsitektur *interbase* sebagai berikut:

a. Super Server

Super Server merupakan multi klien, implementasi multi-thread dari proses *server InterBase*. Implementasi ini menggantikan jenis implementasi model *Classic* yang telah digunakan pada versi *InterBase* sebelumnya.

Super Server melayani banyak klien dalam waktu yang sama menggunakan urutan daripada proses server yang terpisah untuk setiap klien. *Multiple thread* membagi akses kepada proses *single server*. Keuntungan dari *Super Server* adalah:

1. Memiliki sebuah proses *server* mengeliminasi kemacetan hasil dari arbitrasi untuk membagi halaman-halaman database dan mengurangi kenaikan yang diperlukan untuk *multiple* proses memulai dan permintaan *database*.

2. *Super Server* meningkatkan kinerja interaksi pesan karena pemanggilan *sharing library* selalu lebih cepat daripada *interproses* permintaan komunikasi pada proses server.
3. *Super Server* meningkatkan integritas database karena hanya satu proses server yang telah ditulis mengakses database, daripada satu proses untuk setiap klien.
4. *Super Server* mengijinkan dalam mengoleksi *statistic database* dan informasi user yang dapat digunakan oleh tool *InterBase* untuk kinerja pengawasan dan tugas administratif.
5. *Super Server* bernilai lebih efektif ketimbang arsitektur *Classic*. Semua sistem operasi memiliki batas nilai dari operasi sistem operasi yang dapat berjalan secara bersama-sama. *Super Server* mengijinkan untuk angka dari database yang telah ditetapkan meneruskan menjadi digandakan angka yang besar secara potensial dari koneksi database berbarengan.

b. Classic

Jenis arsitektur lainnya yaitu *Classic*, merupakan design dari *InterBase versi 4.0* dan sebelumnya, adalah didasarkan pada proses. Untuk setiap koneksi klien, proses server yang terpisah dimulai dengan mengeksekusi *engine database* dan setiap proses server memiliki suatu *cache database* yang didedikasikan. Proses server ditentang untuk mengakses database, sehingga subsistem *Lock Manager* diperlukan untuk mengarbitrasi dan mengsinkronisasi akses database berbarengan diantara proses-proses.

Server *InterBase Classic* berjalan berdasar permintaan multiple proses. Sewaktu klien berusaha mengkoneksi ke suatu database *Interbase,* suatu *executable* dari *gds_inet_server* berjalan dan sisanya didedikasikan kepada koneksi klien tersebut selama durasi koneksi. *Inisiator* dari *gdnet_inet_server* adalah *inetd*, merupakan proses pelayanan pengalihan kunci dari *UNIX*. Ia memiliki sebuah konfigurasi file *(/etc/inetd.conf*) yang mengasosiasikan pelayanan dengan *executable* yang dipakai untuk menerima koneksi. Ketika inetd menerima sebuah koneksi yang diminta untuk sebuah servis, ia akan mencari program yang tepat pada */etc/inetd.conf*, kemudian mengeksekusinya dan mentransfer koneksi jaringan ke program servis. Ketika klien memutuskan untuk *disconnect, gds_inet_server* menutup koneksi ke database dan file-file lainnya tersebut. Sewaktu tidak ada klien yang terkoneksi ke database, seharusnya tidak terdapat invokasi dari *gds_inet_server* yang sedang berjalan.

*Lock Management***,** diambil alih oleh proses lainnya yaitu *gds_lock_mgr***.** Program ini akan memulai ketika klien kedua masuk untuk diberikan database. Pekerjaan dari *lock manager* adalah untuk pelayanan, sebagai *traffic cop.* Ia memberikan kunci sumber database kepada klien. Ia juga meminta klien melepaskan kunci pada suatu sumber sewaktu *resource* tersebut sedang diminta oleh klien lain. Ia masih berjalan meskipun klien terakhir *disconnect*. Untuk waktu berikutnya suatu klien melakukan koneksi, akan terhindar dari kelebihan kelalaian dari proses *lock manager*.

Gds_lock_mgr memproses komunikasi kepada setiap proses klien dengan penggunaan area *share memory* dan sebuah mekanisme sinyal menggunakan sinyal *Posix,* yaitu *SIGUSR1* dan *SIGUSR2*. Sinyal-sinyal tersebut ditangkap dalam rutin sinyal handling pada *libgdslib,a,* dan untuk alasan ini aplikasi pengguna seharusnya tidak menampilkan sinyal handling atau memodifikasi *signal mask.* Seluruh aplikasi klien decompile dengan menggunakan *ligds,a.* Tidak ada perubahan yang diperlukan, hanya *option* hubungan yang berbeda.

Setiap *instance* dari *gdnet_inet_server* menyimpan *cache* dari halaman-halaman database pada setiap ruang memorinya, digunakan untuk menghasilkan beberapa duplikasi dari data *cache* melalui sistem. Selama penggunaan *resource* per klien lebih besar pada *Super Server,* model *Classic* secara keseluruhan lebih sedikit menggunakan *resource* ketika angka dari koneksi yang dilakukan berbarengan jumlahnya sedikit.

Arsitektur *Classic* mengijinkan pemrosesan dari aplikasi untuk melakukan *I/O* pada file-file database secara langsung, mengingat arsitektur *Super Server* membutuhkan aplikasi untuk meminta operasi *I/O ibserver* dari *proxy*, menggunakan metode jaringan. Metode akses *local* lebih cepat daripada metode akses jaringan, tetapi hanya berguna dengan aplikasi-aplikasi yang berjalan pada *host* yang sama sebagai database.

Dalam memonitor database, pada model *Classic*, informasi panggilan dari koneksi database selalu melaporkan tepatnya satu koneksi, tidak terpengaruh terhadap berapa banyaknya klien yang melakukan koneksi kepada server tersebut. Hal ini terjadi karena setiap koneksi klien memiliki proses *gds_inet_server* masing-masing. Hanya pad model *Super Server* saja yang memiliki kemampuan untuk melaporkan seluruh koneksi klien pada server.

Mengenai sekuritas, dalam hal model *Classic* bekerja percampuran antara local dan remote klien yang berjalan sebagai userID yang berbeda, server yang dapat diesekusi *gdnet_inet_server* dan *gds_lock_mgr* harus berjalan sebagai *root*. Prosesnya harus berjalan dengan *userID* sebenarnya dari root untuk mengeset *userID* efektif mereka, yaitu *userID* klien tersebut. Dalam hal ini *Lock Manager* harus memiliki hak sebagai *super user* untuk mengirim sinyal kepada proses tersebut.