2. LANDASAN TEORI

Di dalam bab ini di bahas tentang landasan teori yang di gunakan dalam pembuatan Tugas Akhir. Secara singkat di jelaskan mengenai sistem informasi, *database*, *SQL Server*, dan semua jenis program yang mendukung pengerjaan Tugas Akhir ini.

2.1. Sistem Informasi

Secara umum sistem informasi dapat di definisikan sebagai sekumpulan komponen pembentuk sistem yang mempunyai keterkaitan antara satu komponen dengan komponen lainnya yang bertujuan menghasilkan suatu informasi dalam suatu bidang tertentu. Dalam sistem informasi di perlukannya klasifikasi alur informasi, hal ini di sebabkan keanekaragaman kebutuhan akan suatu informasi oleh pengguna informasi. Kriteria dari sistem informasi antara lain, fleksibel, efektif dan efisien. (Kadir,(2003))

Dengan semakin berkembangnya era teknologi, menyebabkan era perdagangan pun terkena dampaknya. Dengan begitu, komputer sangat di perlukan untuk membantu, memudahkan, mempercepat proses perdagangan tersebut. Pada akhirnya, para ahli menciptakan sebuah trobosan baru agar dapat lebih membantu para pengusaha untuk menjalankan perusahaan mereka. Maka muncul yang namanya sistem informasi. Sistem informasi manajemen menerapkan penyajian sistem informasi yang cepat dan akurat, hal itu dapat membantu seorang manajer dalam pengambilan keputusan.

Dalam pelaksanaannya, sistem informasi ini akan sangat membantu misalnya dengan adanya data yang sangat banyak, perusahaan tidak perlu lagi melakukan sistem administrasi secara manual. Dengan sistem informasi ini perusahaan dapat melakukan sistem administrasi secara cepat, tepat dan akurat. Kelebihan lain yang di dapat dari adanya sistem informasi ini adalah dapat menghemat waktu dan biaya di bandingkan dengan pengerjaan secara manual.

Tujuan sistem informasi manajemen antara lain :

- Membantu bagian administrasi dalam pelaksanaan tugasnya.
- Menghemat waktu dan biaya
- Megurangi tingkat kesalahan *human error*.
- Membantu atasan dalam pengambilan keputusan.

2.2. *Database*

Database adalah sebuah rangkaian data yang saling berhubungan dan terstruktur. Data yang telah di simpan berfungsi sebagai sumber dalam pengolahan informasi.

Database Management System (DBMS) adalah perangkat lunak yang memungkinkan para pengguna dapat membuat, memelihara, mengontrol, dan mengakses *database* dengan cara yang praktis dan efisien. DBMS menerima permintaan data untuk program aplikasi dan memberikan perintah pada sistem operasi untuk men-*transfer* data secara tepat. Untuk mengelola *database* di perlukan perangkat lunak yang disebut DBMS (*Database Management System).*

Beberapa kelemahan DBMS : (McLeod, 1996)

- Memerlukan konfigurasi *hardware* dan memori yang besar agar sistem dapat bekerja secara optimal.
- Kompleksitas yang tinggi, membuat para pemakai harus memahami setiap fungsi-fungsi dengan baik agar dapat memperoleh manfaat yang optimal.
- Memerlukan keahlian dan pengetahuan khusus untuk dapat memanfaatkan semua fasilitas.

Beberapa keunggulan DBMS : (McLeod,1996)

- Dapat menghemat biaya karena data di pakai oleh banyak departemen.
- Meningkatkan produktivitas program
- Mengurangi kelebihan data (*redundancy data)*
- Memperoleh data tunggal
- Menggabungkan kumpulan data dari beberapa file.

2.3. *Data Flow Diagram* (DFD)

Pengertian *Data Flow Diagram* (DFD) adalah suatu diagram yang menggunakan notasi - notasi untuk menggambarkan arus dari data sistem, yang penggunaannya sangat membantu untuk memahami sistem secara logika, terstruktur dan jelas.

DFD merupakan alat bantu dalam menggambarkan atau menjelaskan sistem yang sedang berjalan logis.

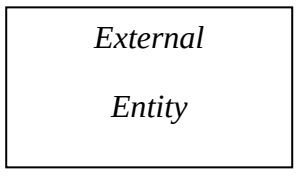
Ada beberapa simbol – simbol yang di gunakan untuk menggambarkan sistem dalam DFD, diantaranya adalah :

- *Data Store*, merupakan simbol yang di gunakan sebagai tempat penyimpanan data yang dapat berupa suatu file atau *database.* Tempat penyimpanan ini dapat di gunakan jika suatu proses perlu menggunakan data tersebut.

Data Store

Gambar 2.1 *Data Store*

- *External Entity*, adalah simbol yang mewakili elemen yang berada di luar sistem, tetapi memiliki hubungan interaksi dengan sistem.

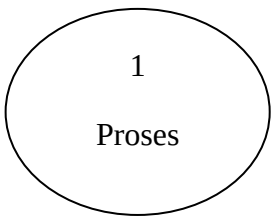


Gambar 2.2 *External Entity*

- *Data Flow*, merupakan simbol yang menunjukkan arah aliran data yang berasal dari satu atau beberapa proses, *data store* atau elemen lingkungan menuju ke satu atau beberapa proses, *data store* atau *environmental element* lainnya. *Data Flow* ini di simbolkan dengan tanda panah dan di beri keterangan di sampingnya yang menunjukkan data apa yang mengalir.

Gambar 2.3 *Data Flow*

- Proses, merupakan simbol yang mewakili kegiatan untuk mengubah suatu data dari suatu bentuk menjadi bentuk yang lain. Proses menerima *input* data dan mengeluarkan *output* data lain yang di proses.



Gambar 2.4 Proses

- *Context Diagram* adalah pemodelan sistem dengan menggunakan DFD melalui beberapa tahap mulai dari *context diagram* hingga ke *level* di mana sistem yang di jabarkan sudah *detail.*

Dalam penggambarannya DFD perlu di berikan penomoran yang jelas pada tiap tahapnya. Misalnya pada DFD level 1, proses di beri nomor 3,4 dan seterusnya. Kemudian pada DFD *level* 2, subproses dari proses nomor 1 di beri nomor 3.1,3.2,3.3 dan seterusnya.

2.4. *Power Designer*

Power designer terdiri dari 4 bagian, yaitu : *AppModeler, Data Architect, MetaWorks,* dan *Process Analyst.* Dalam pembuatan aplikasi ini hanya menggunakan dua buah bagian dari *Power Designer* yaitu *Data Architect* dan *ProcessAnalyst. ProcessAnalyst* menggambarkan sistem kerja sebuah *physical* model, sedangkan *Data Architect* digunakan sebagai alat bantu dalam perancangan *Conceptual Data Model* (CDM). Di mana dari CDM *(Conceptual Data Model)* ini dapat meng-*generate* sebuah *Physical Data Model* (PDM). *Conceptual Data Model* sendiri dapat mempresentasikan struktur logik dari sebuah basis data secara keseluruhan. Sebuah *conseptual* model seringkali mengandung objek data yang belum di implementasikan pada basis data secara nyata. Sebuah *conceptual* model memberikan *enterprise* atau aktivitas bisnis. Sebuah *Physical Data Model* (PDM) menspesifikasikan implementasi fisik dari sebuah basis data.

2.5. *Structure Query Language* (SQL)

Structure Query Language (SQL) adalah bahasa terstruktur yang di gunakan untuk *query,* meng-*update,* dan mengolah relasi antara *database.* Perintah-perintah SQL di gunakan untuk melakukan pekerjaan – pekerjaan tertentu pada *database* seperti *update* data atau untuk mengambil data dari *database.* Di bawah ini akan di jelaskan perintah - perintah yang akan di gunakan dalam SQL.

2.5.1. Perintah *Select*

Perintah *select* di gunakan untuk mengambil dan menampilkan data dari *database* sesuai dengan syarat - syarat yang di tentukan. Berikut ini adalah format perintah *select* yang umum di gunakan.

```
SELECT coloumn1 [ ,coloumn2,etc]

FROM tablename

[WHERE condition];

Keterangan :[]=optional
```

Coloumn menunjukkan nama kolom dari table yang ingin di ambil. *Coloumn* dapat lebih dari satu atau menggunakan '*' untuk mengambil semua kolom. *Tablename* menunjukkan nama table yang ingin di gunakan. Klausa *where* menyaring data berdasarkan kondisi yang setelah kata *where.* Operator – operator yang di gunakan dalam klausa *where* antara lain:

Tabel 2.1. Operator Untuk Perintah SQL

LAMBANG	ARTI
=	Sama Dengan
>	Lebih besar dari
<	Lebih kecil dari
>=	Lebih besar sama dengan
<=	Lebihn kecil sama dengan
<>	Tidak sama dengan
LIKE	(dijelaskan dibawah)

Operator *LIKE* dapat menyeleksi data sesuai dengan *string* yang di inginkan. Simbol '%' di gunakan untuk mencocokkan karakter yang muncul sebelum atau sesudah karakter yang di inginkan. Contohnya adalah sebagai berikut :

```
SELECT nama_depan, alamat

FROM mahasiswa

WHERE nama_depan LIKE 'Yu%';
```

Perintah SQL di atas memiliki *output* semua nama depan yang di awali dengan 'Yu'. Simbol '%' dapat pula di letakkan di depan atau di belakang.

Selain seperti yang di jabarkan di atas, perintah *select* juga memiliki format lain yang lebih lengkap yang mendukung fungsi agregasi, yaitu :

```
SELECT [ALL |   DISTINC] coloumn 1[,coloumn2]
FROM table1[,table 2]
[WHERE”conditions”]
[GROUP BY “coloumn-list”]
[HAVING”conditions]
```

Table 2.2. Fungsi Agregasi

Nama Fungsi	Kegunaan
SUM( )	Untuk menghitung jumlah nilai pada suatu kolom
AVG ( )	Untuk menghitung nilai rata – rata pada suatu kolom
MAX ( )	Untuk mencari nilai maksimum pada suatu kolom
MIN ( )	Untuk mencari nilai minimal pada suatu kolom
COUNT ( )	Unuk menghitung jumlah baris pada suatu kolom

Keyword ALL di gunakan untuk mengambil semua data sedangkan *DISTINC* di gunakan untuk mengambil data yang unik di mana baris yang sama akan di jadikan satu baris saja.

2.5.1.1. *Klausa Group By*

Klausa Group By akan mengumpulkan baris data dari table sesuai dengan kolom yang dispesifikasikan dan memperkenankan adalah fungsi agregasi pada satu atau lebih kolom. Untuk lebih jelas dapat dilihat pada contoh berikut :

```
SELECT MAX ( gaji ), department
FROM pegawai
GROUP BY department;
```

Query ini akan mengambil dan menampilkan gaji maksimum dari pegawai dari tiap departmen dan di kelompokkan berdasarkan nama departmennya.

2.5.1.2. *Klausa Order By*

Klausa Order By merupakan klausa *optional* yang di gunakan jika kita ingin menampilkan hasil dari *query* secara urut, baik secara *ascending* (ASC) maupun secara *descending* (DESC), sesuai dengan kolom yang di spesifikasikan pada *order by*.

Contoh penggunaannya jika kita ingin mengurutkan data pegawai berdasarkan nama depannya secara *ascending* adalah sebagai berikut :

```
SELECT kode_pegawai, nama_depan, departemen, kota
FROM pegawai
WHERE departemen = ‘ Penjualan ‘
ORDER BY nama_depan ASC;
```

2.5.2. Perintah *Insert*

Perintah ini di gunakan untuk memasukkan atau manambah baris data ke dalam sebuah tabel. Format penulisan perintah *insert* adalah sebagai berikut :

```
INSERT INTO tablename (first_column,…,last_column)
VALUES (first_value,…,last_value);
```

Parameter setelah *tablename* adalah daftar kolom – kolom yang di pisahkan dengan koma, di ikuti dengan kata *vaues* lalu di ikuti lagi dengan daftar nilai yang urutannya sesuai dengan daftar kolom sebelum kata *values.* Untuk nilai yang bernilai *string* harus di apit dengan tanda petik satu sedangkan untuk yang bernilai *integer* tidak perlu di apit dengan tanda petik satu. Contohnya adalah sebagai berikut :

```
INSERT INTO karyawan (nama_depan,nama_belakang,umur,kota)
VALUES (‘Bagus’, ‘Wahyudi’,23, ‘Surabaya’)
```

Query di atas akan menambahkan baris baru pada tabel karyawan dengan *field* nama_depan bernilai ‘Bagus’, *field* nama_belakang bernilai ‘Wahyudi’, *field* umur bernilai 23 dan *field* kota bernilai ‘Surabaya’.

2.5.3. Perintah *Update*

Perintah *update* d igunakan untuk mengubah data yang sudah ada berdasarkan kondisi tertentu yang di jabarkan oleh *keyword where.* Format perintah lengkapnya adalah sebagai berikut:

```
UPDATE tablename

SET columnname =newvalue [,nextcolumn= newvalue2..]

WHERE columnname <OPERATOR> value

[and  or column “<OPERATOR>”value” ];

Keterangan : [] = optional

Contoh :

UPDATE pegawai

SET umur = 23

WHERE nama_depan = ‘Bagus’;
```

2.5.4. Perintah *Delete*

Perintah *delete* di gunakan untuk menghapus baris dari suatu tabel. Formatnya adalah sebagai berikut :

```
DELETE FROM tablename

WHERE columnname <OPERATOR>value[and or
column<operator>value]

Keterangan [] = optional

Contoh penggunaannya :
```

```
DELETE FROM pegawai

WHERE nama_belakang = ‘Bagus‘ ;
```

Contoh diatas akan menghapus baris dari tabel ‘pegawai’ yang *field* ‘nama_belakang’-nya bernilai ‘Bagus’. Untuk menghapus seluruh baris dari suatu tabel, kosongkan klausa *where.*

2.6. Entity Relationship Diagram (ERD)

Entity Relationship Diagram (ERD) memiliki pengertian “A graphical technique for partraying a data base schema “ (Roomney, (2003)).

ERD adalah peralatan pembuatan model data yang paling fleksibel dan dapat di adaptasi untuk berbagai pendekatan yang mungkin di ikuti oleh perusahaan dalam pengembangan sistem. ERD menggambarkan relasi atau hubungan antar entitas yang ada, di mana terdapat 2 jenis hubungan, yaitu :

a. *Non-Obligatory* adalah bila tidak semua anggota dari suatu *entity* harus berpartisipasi atau memiliki hubungan dengan *entity* yang lain.

Gambar 2.5. *Non – Obligatory*

b. *Obligatory* adalah kebalikan dari *non – obligatory* yaitu, bila semua anggota dari suatu *entity* harus berpartisipasi atau memiliki hubungn dengan *entity* yang lain:

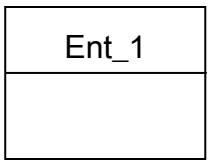
Gambar 2.6. *Obligatory*

Keterangan gambar : gambar menunjukkan bahwa semua anggota dari *entity2* harus berelasi dengan anggota dari *entity* 1, dengan memberi simbol ” “ di depan *entity* 2. Pada *entity* 1 terhadap *entity* 2 di beri simbol ” 0 “ di depan *entity* 1, berarti anggota dari *entity* 1 tidak harus berelasi seluruhnya dengan anggota dari *entity* 2.

Dalam menggambar ERD, ada beberapa komponen yang perlu di perhatikan, antara lain : (Hoffer,1999).

a. *Entity*

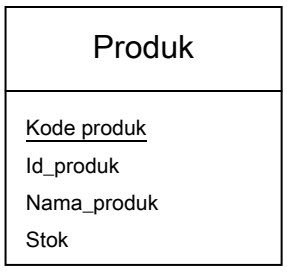
Entity di definisikan sebagai sesuatu yang mudah di identifikasikan. Sebuah *entity* dapat berupa objek, tempat, orang, konsep atau aktivitas. Pada teknik penggambaran, *entity* di gambarkan dengan kotak segiempat. Setiap kotak di beri *label* berupa kata benda. Simbol *entity* dapat di lihat pada Gambar 2.7.



Gambar 2.7. *Entity*

b. *Attribute*

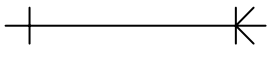
Attribute adalah karateristik dari *entity*. Identifikasi dan deskripsi dari *entity* di jelaskan oleh atribut – atibutnya. Sebuah atribut di definisikan sebagai penjelasan – penjelasan dari *entity* yang lain. Selain itu, atribut juga merupakan sifat – sifat dari sebuah *entity*. Sebagai contoh, *entity Produk* mempunyai atribut kode_produk, id_produk, dan atribut lainnya. Contoh atribut dapat dilihat pada Gambar 2.8.



Gambar 2.8. *Attribute*

c. Relation

Relasi adalah penghubung antara suatu *entity* yang lain dan merupakan bagian yang sangat penting dalam mendesain *database*. Simbol relasi dapat dilihat dari Gambar 2.9.

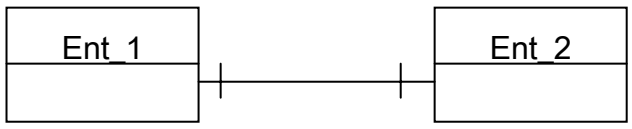


Gambar 2.9. *Relation*

Bentuk relasional yang dapat terjadi pada model *database* adalah :

a. *One-to-One-Relationship*

Pada bentuk relasi *One-to-One*, satu anggota *entity* memiliki hubungan dengan satu anggota *entity* pada kelas yang berbeda. Simbol relasi *One-to-One* dapat di lihat dari Gambar 2.10.



Gambar 2.10.Relasi *One-to-One*

b. *One-to-many-Relationship*

Pada bentuk relasi *One-to-many*, satu anggota *entity* bisa memiliki hubungan dengan beberapa anggota *entity* pada kelas yang berbeda. Simbol relasi *One-to-Many* dapat di lihat pada Gambar 2.11.

Gambar 2.11. Relasi *One-to-Many*

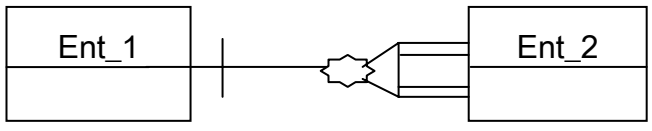
c. *Many-to-many-Relationship*

Pada bentuk relasi *Many-to-many*, beberapa anggota *entity* dapat memiliki hubungan dengan beberapa anggota *entity* lainnya. Simbol relasi *Many-to-many* dapat di lihat dalam Gambar 2.12.

Gambar 2.12. Relasi *Many-to-many*

d. Dependency

Dependency melambangkan ketergantungan dengan *entity* yang bersangkutan. Gambar 2.13. menunjukkan bahwa Entity 1 dengan Entity 2 merupakan tabel yang saling bergantung. Simbol *dependency* dapat di lihat dari Gambar 2.13.



Gambar 2.13. *Dependency*

2.7. *Borland Delphi 7*

Salah satu aplikasi yang sering di kembangkan menggunakan *Borland Delphi 7* adalah aplikasi *database.* Dalam membangun aplikasi *database*, *Delphi 7* memiliki beberapa metode. Salah satu metode yang di gunakan dalam pembuatan aplikasi ini adalah ADO (*ActiveX Data Object*), komponen *database* ADO adalah komponen *database* yang di buat oleh *Microsoft.* Dengan komponen ini, dapat di buat koneksi maupun memanipulasi data pada *database* (Mangkulo,2004).

Borland Delphi 7 merupakan salah satu program pengembangan aplikasi produksi dari Borland. *Delphi 7* adalah bahasa pemrograman visual yang sudah terkenal akan keandalannya, di mana dapat dengan mudah di gunakan untuk mengatur tampilan dari program aplikasi, sehingga dapat lebih difokuskan ke dalam pembuatan program aplikasi tersebut.

Secara visual *Borland Delphi 7* di bagi menjadi beberapa bagian, yaitu : menu utama, *speed bar, component pallet,* jendela *form, object inspector,* dan *object treeview. Form* merupakan tampilan berbentuk jendela yang memungkinkan *user* melakukan interaksi dengan aplikasi. *Component pallete* berisi komponen – komponen yang telah di sediakan *Delphi* untuk memudahkan pembuatan program.

2.8. Laporan Laba Rugi (*Income Statement*)

Laporan laba rugi adalah laporan yang biasa di buat perusahaan di akhir periode, berisi tentang pendapatan dan pengeluaran.(McGraw. (2008))

Hal – hal yang dapat di lihat dalam laporan laba rugi :

1. Biaya operasional, terdiri dari biaya administrasi, biaya penjualan dan biaya lain – lain. Biaya – biaya ini mendukung kegiatan non produksi seperti biaya pemasaran, gaji karyawan dan biaya lainnya.
2. Laba kotor, mengukur langsung laba dari penjualan atau jumlah laba yang di peroleh perusahaan yang merupakan hasil pengurangan antara penjualan dan pembelian.
3. Laba bersih, laba yang di dapat setelah di kurangi biaya operasional.
4. Penjualan bersih, merupakan selisih dari penjualan kotor perusahaan dengan pengembalian penjualan atau *discount*. Jika ada barang yang di kembalikan, nilainya harus di kurangi dari penjualan kotor pada periode tersebut. Demikian pula sama hal nya jika di lakukan penjualan kredit, perusahaan

akan menawarkan harga yang lebih rendah untuk pembayaran yang lebih cepat atau di berikan diskon. Jika pelanggan menerima diskon, nilai uang dari diskon harus di kurangi dari penjualan kotor.

5. Penjualan kotor adalah kuantitas barang yang terjual di kali harga jual barang. Penjualan ini dapat berupa penjualan tunai dan penjualan kredit. Penjualan ini merupakan transaksi dari penjualan barang dan penjualan jasa.

Contoh Laporan Keuangan (Laba Rugi) :

PT. ABC

Laporan Laba Rugi

(dalam rupiah)

Penjualan	1.000.000
Pembelian	(300.000)
Laba kotor	700.000
Biaya Operasional	(50.000)
Biaya penjualan	(50.000)
Biaya administrasi dan biaya lain – lain	(30.000)
Total biaya Operasional	130.000
Laba bersih	570.000

Gambar 2.14. Laporan Laba Rugi

2.9. Metode FIFO (*First In First Out*)

Secara umum FIFO adalah akronim untuk Dalam Pertama, Keluar Pertama, yang abstak dalam cara mengatur dan manipulasi data yang relatif terhadap waktu dan prioritas. Dalam arti lebih luas, yang abstak LIFO atau *Last-In-First-Out* adalah kebalikan dari FIFO. (McGraw. (2008))

- Metode FIFO mengasumsikan bahwa barang yang pertama kali di beli adalah barang yang pertama kali di jual.
- COGS (*Cost Of Goods Sold*) dalam metode FIFO didapat dari harga barang yang pertama kali dibeli.
- Jika ingin mengetahui EI (*Ending Inventory*) yang dimiliki dapat dihitung dari pembelian barang yang terakhir.

Contoh kasus :

Tanggal	Unit	Unit Cost	Total Cost
1 Januari	80	15	1200
15 Maret	60	16	960
20 Juni	100	17.5	1750
25 Oktober	90	18	1620

EI = 110 Unit

Cara Perhitungan menggunakan metode FIFO :

EI = 90 x 18 = 1.620

20 x 17,5 = 350 +

1.970

COGS = 5.530 – 1.970

= 3.560

Jika di analisa secara sistematis :

Ending Inventory = (Beginning Inventory + Purchased) – COGS

Goods available for sale

Cost Of Goods Purchased = NetPurchase + Freight in

(Purchase – Purchase Discount – Purchase Return & Allowence)

NetSale = Sale – SalesDiscount – Sales Return & Allowence

Income Statement

Sales Revenue		XXX
COGS		XXX +
Gross Profit	XXX	
Expense	XXX -	
NetIncome		XXX