BAB 2

LANDASAN TEORI

2.1 Teori-teori Dasar/Umum

2.1.1 Pengertian Data

Pengertian data menurut Considine, B., Parkes, A., Olesen, K., Speet, D., & Lee, M. (2010), data adalah fakta mentah yang berkaitan dengan atau menjelaskan suatu peristiwa.

Menurut Pearlson, K.E. & Saunders, C.S. (2010), data adalah seperangkat spesifik, fakta-fakta objektif atau pengamatan, seperti "persediaan mengandung 45 unit". Berdiri sendiri, fakta-fakta tersebut telah ada makna intrinsik, tetapi dapat dengan mudah ditangkap, dikirim dan disimpan secara elektronik.

Menurut O'Brien, J.A. & Marakas, G.M. (2010) data adalah bentuk jamak dari datum, meskipun biasanya mewakili bentuk tunggal maupun jamak. Data adalah fakta mentah atau pengamatan, biasanya fenomena fisik atau transaksi bisnis.

Berdasarkan pengertian data di atas dapat disimpulkan bahwa data adalah kumpulan fakta mentah yang belum diolah.

2.1.2 Pengertian Basis Data

Pengertian basis data menurut Connolly, T.M. & Begg, C.E. (2010), basis data adalah koleksi bersama data secara logis terkait dan deskripsi, yang dirancang untuk memenuhi kebutuhan informasi dari suatu organisasi.

Menurut Considine, B., et al. (2010), basis data adalah sebuah struktur komputerisasi bersama yang menangkap, menyimpan dan menghubungkan data.

Menurut Satzinger, J.W., Jackson, R.B. & Burd, S.D. (2010), basis data adalah koleksi terpadu dari data yang tersimpan yang terpusat dikelola dan dikendalikan.

Berdasarkan perngertian basis data di atas dapat disimpulkan bahwa basis data adalah kumpulan data-data teroganisir yang telah disimpan.

2.1.3 Pengertian Perancangan Basis Data

Pengertian perancangan basis data menurut Connolly, T.M., et al. (2010), perancangan basis data adalah proses menciptakan desain yang akan mendukung rumusan misi dan tujuan misi perusahaan untuk sistem basis data yang diperlukan.

Menurut Hegazi, M. O. A. (2014), perancangan basis data adalah tugas sulit yang membutuhkan pemahaman penuh dari aplikasi dan menerjemahkan persyaratan desain ke dalam model konseptual. Tujuan dari perancangan basis data adalah untuk memenuhi kebutuhan isi informasi, memberikan struktur alami dan mudah untuk memahami informasi dan pengolahan dukungan persyaratan dan tujuan setiap kinerja.

2.1.4 *Database Management System* (DBMS)

2.1.4.1 Pengertian *Database Management System* DBMS

Pengertian *database management system* (DBMS) menurut Connolly, T.M., et al. (2010), *database management system* (DBMS) adalah sistem perangkat lunak yang memungkinkan pengguna untuk mendefinisikan, membuat, memelihara, dan mengontrol akses ke database.

Menurut Laudon, K.C. & Laudon, Jane.P. (2012), *database management system* (DBMS) adalah perangkat lunak yang memungkinkan sebuah organisasi untuk memusatkan data, mengelola secara efisien, dan memberikan akses ke data yang disimpan oleh program aplikasi.

Menurut Hall, J.A. (2011), *database management system* (DBMS) adalah perangkat lunak sistem khusus yang diprogram untuk mengetahui elemen data setiap pengguna berwenang untuk mengakses.

2.1.4.2 Fasilitas DBMS

Menurut Connolly, T.M., et al. (2010), Secara khusus, DMBS menyediakan fasilitas sebagai berikut:

- Memungkinkan pengguna untuk menentukan database, biasanya melalui *Data Definition Language* (DDL). DDL memungkinkan pengguna untuk menentukan tipe data dan struktur dan kendala pada data yang akan disimpan dalam database.
- Memungkinkan pengguna untuk melakukan operasi *insert, update, delete,* dan mengambil data dari *database*, biasanya melalui *Data Manipulation Language* (DML).
- Menyediakan kontrol akses ke database, meliputi :
  - Sistem keamanan

    Mencegah pengguna yang tidak sah mengakses database.
  - Sistem integritas

    Mempertahankan konsistensi data yang disimpan.
  - Sistem kontrol *concurrency*

    Memungkinkan akses bersama database.
  - Sistem kontrol *recovery*, yang

    Mengembalikan database ke keadaan yang konsisten sebelumnya setelah kegagalan *hardware* atau *software*.
  - User-accessible catalog

    Berisi deskripsi dari data dalam database.

2.1.4.3 Komponen Lingkungan DBMS

Menurut Connolly, T.M., et al. (2010), ada 5 komponen dalam lingkungan DBMS yaitu:

- *Hardware* (Perangkat Keras)

  DBMS dan aplikasi memerlukan perangkat keras untuk menjalankan. Perangkat keras dapat berkisar dari komputer pribadi tunggal ke *mainframe* tunggal dan ke

jaringan komputer. Perangkat keras tertentu tergantung pada kebutuhan organisasi dan DBMS yang digunakan. Beberapa DBMS hanya berjalan pada perangkat keras atau operasi tertentu sistem, sementara yang lain berjalan pada berbagai hardware dan sistem operasi. Sebuah DBMS membutuhkan jumlah minimal memori utama dan ruang *disk* untuk menjalankan, tetapi konfigurasi minimum ini mungkin tidak selalu memberikan kinerja yang dapat diterima.

- *Software* (Perangkat Lunak)

  Komponen perangkat lunak terdiri dari perangkat lunak DBMS itu sendiri dan program aplikasi, bersama dengan sistem operasi, termasuk perangkat lunak jaringan jika DBMS digunakan melalui jaringan. Biasanya, program aplikasi yang ditulis dalam bahasa pemrograman generasi ketiga (3GL), seperti 'C', C ++, Java, Visual Basic, COBOL, Fortran, Ada, atau Pascal, atau menggunakan bahasa generasi keempat (4GL), seperti SQL, tertanam dalam bahasa generasi ketiga. Target DBMS mungkin memiliki alat generasi keempat sendiri yang memungkinkan perkembangan pesat dari aplikasi melalui penyediaan bahasa query non-prosedural, laporan generator, membentuk generator, generator grafis, dan generator aplikasi. Penggunaan alat-generasi keempat dapat meningkatkan produktivitas secara signifikan dan menghasilkan program yang lebih mudah untuk pemeliharaan.

- *Data*

  Komponen yang paling penting dari lingkungan DBMS, khususnya dari sudut pandang pengguna akhir.

- *Procedure*

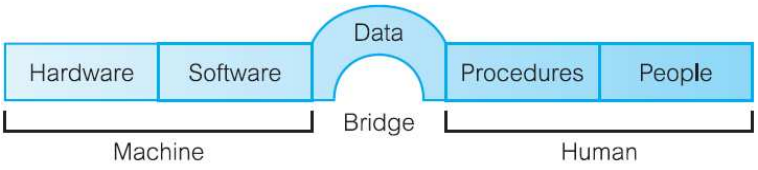
  Prosedur mengacu pada instruksi dan aturan yang mengatur desain dan penggunaan *database*. Para pengguna sistem dan staf yang mengelola *database*

memerlukan prosedur bagaimana menggunakan atau menjalankan sistem didokumentasikan. Ini mungkin terdiri dari petunjuk tentang cara untuk:

- Log on ke DBMS
- Menggunakan fasilitas DBMS tertentu atau program aplikasi
- Memulai dan menghentikan DBMS
- Membuat salinan cadangan dari database
- Menangani kegagalan perangkat keras atau perangkat lunak. Ini mungkin termasuk prosedur bagaimana mengidentifikasi komponen gagal, bagaimana memperbaiki komponen gagal dan, setelah perbaikan kesalahan, bagaimana memulihkan *database*
- Mengubah struktur tabel, mengatur ulang *database* di beberapa *disk*, meningkatkan kinerja, atau arsip data ke penyimpanan sekunder.

- *People* (Manusia)

  Komponen terakhir yang terlibat dalam sistem.



Gambar 2.1 Lingkungan DBMS (Connolly, T.M., et al. (2010))

2.1.4.4 Fungsi DBMS

Menurut Connolly, T.M., et al. (2010), fungsi DBMS yaitu:

- *Data storage, retrieval, and update*

  Sebuah DBMS harus melengkapi pengguna dengan kemampuan untuk menyimpan, mengambil, dan *update* data dalam *database*.

- *A user-accessible catalog*

  Sebuah DBMS harus memberikan katalog di mana deskripsi item data yang disimpan dan yang diakses oleh pengguna.

- *Transaction support*

  Sebuah DBMS harus memberikan mekanisme yang akan memastikan baik bahwa semua *update* yang berhubungan dengan transaksi yang diberikan dibuat atau bahwa tak satu pun dari mereka dibuat.

- *Concurrency control services*

  Sebuah DBMS harus menyediakan mekanisme untuk memastikan bahwa *database* diperbarui dengan benar ketika beberapa pengguna memperbarui *database* secara bersamaan.

- *Recovery services*

  Sebuah DBMS harus memberikan mekanisme untuk memulihkan database dalam hal database rusak dengan cara apapun.

- *Authorization services*

  Sebuah DBMS harus menyediakan mekanisme untuk memastikan bahwa hanya pengguna yang berwenang dapat mengakses *database*.

- *Support for data communication*

  Sebuah DBMS harus mampu mengintegrasikan dengan software komunikasi.

- *Integrity services*

  Sebuah DBMS harus menyediakan sarana untuk memastikan bahwa kedua data dalam *database* dan perubahan pada data mengikuti aturan-aturan tertentu.

- *Services to promote data independence*

  Sebuah DBMS harus mencakup fasilitas untuk mendukung kemandirian program dari struktur aktual dari *database*.

- *Utility services*

  Sebuah DBMS harus menyediakan satu set layanan utilitas.

2.1.4.5 Keuntungan DBMS

Menurut Connolly, T.M., et al. (2010), keuntungan DBMS yaitu:

- Pengendalian redundansi data

  Pendekatan *database* mencoba untuk menghilangkan redundansi dengan mengintegrasikan file sehingga beberapa salinan data yang sama tidak disimpan. Namun, pendekatan *database* tidak menghilangkan redundansi sepenuhnya, namun mengendalikan jumlah redundansi yang melekat dalam *database*.

- Konsistensi data

  Dengan menghilangkan atau mengontrol redundansi, dapat mengurangi risiko inkonsistensi yang terjadi. Jika item data disimpan hanya sekali dalam *database*, setiap update untuk nilainya harus dilakukan hanya sekali dan nilai baru tersedia segera untuk semua pengguna. Jika item data disimpan lebih dari sekali dan sistem menyadari hal ini, sistem dapat memastikan bahwa semua salinan item disimpan konsisten.

- Informasi lebih lanjut dari jumlah data yang sama

  Dengan integrasi data operasional, dimungkinkan bagi organisasi untuk memperoleh informasi tambahan dari data yang sama.

- Berbagi data

  Biasanya, file yang dimiliki oleh orang atau departemen yang menggunakannya. Di sisi lain, database milik seluruh organisasi dapat digunakan bersama oleh semua pengguna resmi. Dengan cara ini, lebih banyak pengguna berbagi lebih dari data.

- Peningkatan integritas data

  Integritas *database* mengacu pada validitas dan konsistensi data yang tersimpan. Integritas biasanya dinyatakan dalam bentuk kendala(constraint), yaitu aturan konsistensi bahwa *database* tidak diizinkan untuk dilanggar. Kendala(constraint) mungkin berlaku untuk item data dalam rekor tunggal atau mereka mungkin berlaku untuk hubungan antara catatan.

- Peningkatan keamanan

  Keamanan *database* adalah perlindungan dari *database* dari pengguna yang tidak sah. Tanpa langkah-langkah keamanan yang sesuai, integrasi membuat data lebih rentan daripada sistem berbasis file. Namun, integrasi memungkinkan DBA untuk mendefinisikan dan DBMS untuk menegakkan keamanan *database*. Hal ini memungkinkan pengambilan nama pengguna dan *password* untuk mengidentifikasi orang-orang yang berwenang untuk menggunakan database.

- Penegakan standar

  Integrasi memungkinkan DBA untuk mendefinisikan dan menegakkan standar yang diperlukan. Ini mungkin termasuk standar departemen, organisasi, nasional, atau internasional untuk hal-hal seperti format data untuk memfasilitasi pertukaran data antara sistem, konvensi penamaan, standar dokumentasi, prosedur *update*, dan aturan akses.

- Skala ekonomi

  Menggabungkan semua data operasional organisasi ke dalam satu *database*, dan menciptakan satu set aplikasi yang bekerja pada satu sumber data dapat menghasilkan penghematan biaya. Dalam hal ini, anggaran yang biasanya akan dialokasikan untuk masing-masing departemen untuk pengembangan dan pemeliharaan

sistem berbasis file yang dapat dikombinasikan, memungkinkan menghasilkan total biaya yang lebih rendah, yang menyebabkan ekonomi skala.

- Keseimbangan persyaratan yang saling bertentangan.
  Setiap pengguna atau departemen memiliki kebutuhan yang mungkin bertentangan dengan kebutuhan pengguna lain. Karena *database* berada di bawah kendali DBA, DBA dapat membuat keputusan tentang desain dan penggunaan operasional *database* yang menyediakan penggunaan terbaik dari sumber daya untuk organisasi secara keseluruhan.
- Peningkatan aksesibilitas data dan respon
  Sebagai akibat dari integrasi, data yang melintasi batas-batas departemen diakses secara langsung ke pengguna akhir. Ini memberikan sebuah sistem dengan berpotensi yang jauh lebih berfungsi, misalnya, digunakan untuk menyediakan layanan yang lebih baik kepada pengguna akhir atau klien organisasi.
- Peningkatan produktivitas
  DBMS menyediakan banyak fungsi standar yang *programmer* biasanya harus tulis dalam aplikasi berbasis file. Pada tingkat dasar, DBMS menyediakan semua *routines* penanganan file tingkat rendah khusus dalam program aplikasi. Penyediaan fungsi-fungsi ini memungkinkan *programmer* untuk berkonsentrasi pada fungsi tertentu yang diperlukan oleh pengguna tanpa harus khawatir tentang tingkat rendah rincian pelaksanaan.
- Peningkatan pemeliharaan melalui independensi data.
  DBMS memisahkan deskripsi data dari aplikasi, sehingga membuat aplikasi kebal terhadap perubahan dalam deskripsi data.

- Peningkatan konkurensi

  Dalam beberapa sistem berbasis file, jika dua atau lebih pengguna yang diizinkan untuk mengakses file yang sama secara bersamaan, adalah mungkin bahwa akses akan mengganggu satu sama lain, yang mengakibatkan hilangnya informasi atau bahkan hilangnya integritas. Banyak DBMS mengelola database akses bersamaan dan memastikan masalah tersebut tidak dapat terjadi.

- Peningkatan layanan *backup* dan *recovery*

  Melindungi data dari kegagalan sistem komputer atau program aplikasi. Melibatkan pengambilan cadangan data pada malam hari. Dalam hal kegagalan selama hari berikutnya, cadangan dipulihkan dan pekerjaan yang telah berlangsung sejak *backup* ini hilang dan harus kembali memasukinya. Sebaliknya, DBMS modern yang menyediakan fasilitas untuk meminimalkan jumlah pengolahan yang hilang setelah kegagalan

2.1.4.6 Kerugian DBMS

Menurut Connolly, T.M., et al. (2010), kerugian DBMS yaitu:

- Kompleksitas

  Penyediaan fungsi yang diharapkan dari DBMS yang baik membuat DBMS menjadi bagian yang sangat kompleks dari perangkat lunak. Desainer *database* dan pengembang, data dan administrator basis data, dan pengguna akhir harus memahami fungsi ini untuk mengambil keuntungan penuh dari itu.

- Ukuran

  Kompleksitas dan luasnya fungsionalitas membuat DBMS bagian yang sangat besar dari perangkat lunak, menempati banyak megabyte ruang disk dan membutuhkan sejumlah besar memori untuk menjalankan efisiensi.

- Biaya DBMS

  Biaya DBMS bervariasi, tergantung pada lingkungan dan fungsi yang disediakan. Sebagai contoh, sebuah DBMS single-user untuk komputer pribadi mungkin hanya US $ 100. Namun, *mainframe* besar DBMS *multi-user* melayani ratusan pengguna bisa sangat mahal, mungkin US $ 100.000 atau bahkan US $ 1.000.000. Ada juga biaya pemeliharaan tahunan berulang, yang biasanya persentase dari harga daftar.

- Biaya perangkat keras tambahan

  Persyaratan penyimpanan disk untuk DBMS dan *database* mungkin memerlukan pembelian ruang penyimpanan tambahan. Selanjutnya, untuk mencapai kinerja yang diperlukan, mungkin perlu untuk membeli mesin yang lebih besar, bahkan mungkin sebuah mesin yang didedikasikan untuk menjalankan DBMS. Pengadaan hasil *hardware* tambahan dalam pengeluaran lebih lanjut.

- Biaya konversi

  Dalam beberapa situasi, biaya DBMS dan perangkat keras tambahan mungkin tidak signifikan dibandingkan dengan biaya konversi aplikasi yang ada untuk dijalankan pada DBMS dan perangkat keras baru. Biaya ini juga termasuk biaya pelatihan staf untuk menggunakan sistem baru, dan mungkin kerja dengan staf ahli untuk membantu dengan konversi dan menjalankan sistem.

- Kinerja

  Biasanya, sistem berbasis file ditulis untuk aplikasi tertentu, seperti faktur. Akibatnya, kinerja secara umum sangat baik. Namun, DBMS ditulis menjadi lebih umum, untuk memenuhi banyak aplikasi bukan hanya satu. Efeknya adalah bahwa beberapa aplikasi tidak dapat berjalan secepat DBMS yang digunakan untuk menjalankan fungsinya.

- Dampak yang lebih besar dari kegagalan
  Sentralisasi sumber daya meningkatkan kerentanan sistem. Karena semua pengguna dan aplikasi bergantung pada ketersediaan DBMS, kegagalan komponen-komponen tertentu dapat membawa operasi berhenti.

2.1.5 Pengertian Program Aplikasi Basis Data

Pengertian program aplikasi basis data menurut Connolly, T.M., et al. (2010), program aplikasi basis data adalah sebuah program komputer yang berinteraksi dengan *database* dengan mengeluarkan permintaan yang sesuai (biasanya sebuah pernyataan SQL untuk DBMS).

2.1.6 *Database Languages*

Menurut Connolly, T.M., et al. (2010), *database languages* adalah sebuah sub-bahasa data terdiri dari dua bagian: *Data Definition Language* (DDL) dan Data Manipulation Language (DML). DDL digunakan untuk menentukan skema *database* dan DML digunakan untuk membaca dan memperbarui *database*. Bahasa ini disebut *sublanguages* data karena mereka tidak termasuk konstruksi untuk semua kebutuhan komputasi seperti pernyataan kondisional atau berulang, yang disediakan oleh bahasa pemrograman tingkat tinggi.

2.1.6.1 *Data Definition Language* (DDL)

Menurut Connolly, T.M., et al. (2010), *data definition language* adalah sebuah bahasa yang mengizinkan DBA atau pengguna untuk mendeskripsikan dan menamai entitas, atribut, dan relasi yang dibutuhkan untuk aplikasi, bersama dengan integritas dan keamanan kendala (*constraint*) terkait.

DDL memungkinkan untuk membuat *database*, menghapus *database*, membuat tabel, mengubah atau menghapus tabel maupun *constraint* yang akan dimasukan ke

dalam *database*, dan membuat maupun menghapus *index*. Fungsi-fungsi dari DDL antara lain:

- *Create Database*

  Proses menciptakan *database* berbeda secara signifikan dari produk ke produk. Dalam sistem multi-user, kewenangan untuk membuat *database* biasanya disediakan untuk DBA. Sintaks dasar:

  CREATE DATABASE DatabaseName

- *Drop Database*

  Proses menghapus *database* yang telah dibuat pada server. Sintaks dasar:

  DROP DATABASE DatabaseName

- *Create Table*

  Membuat struktur tabel untuk relasi dasar untuk berada dalam database. Hal ini dilakukan dengan menggunakan pernyataan CREATE TABLE, yang memiliki sintaks dasar sebagai berikut:

  CREATE TABLE TableName
  (
  {(columnName dataType [NOT NULL] [UNIQUE]
  [DEFAULT defaultOption] [CHECK (searchCondition)]
  [, . . .]}

  [PRIMARY KEY (listOfColumns),]

  {[UNIQUE (listOfColumns)] [, . . .]}

  {[FOREIGN KEY (listOfForeignKeyColumns)
  REFERENCES ParentTableName
  [(listOfCandidateKeyColumns)]
  [MATCH {PARTIAL | FULL}
  [ON UPDATE referentialAction]
  [ON DELETE referentialAction]]
  [, . . .]}

{[CHECK (searchCondition)] [, . . .]}

)

- *Alter Table*

  Standar ISO memberikan pernyataan ALTER TABLE untuk mengubah struktur tabel setelah telah dibuat. Definisi pernyataan ALTER TABLE dalam standar ISO terdiri dari enam pilihan untuk:

  - menambahkan kolom baru ke tabel
  - drop kolom dari tabel
  - menambahkan *table constraint* baru
  - menjatuhkan *table constraint*
  - menetapkan standar untuk kolom
  - menghapus default untuk kolom

  Format dasar dari pernyataan itu adalah:

  ALTER TABLE TableName
  [ADD [COLUMN] columnName dataType [NOT NULL] [UNIQUE] [DEFAULT defaultOption]
  [CHECK (searchCondition)]]

  [DROP [COLUMN] columnName
  [RESTRICT | CASCADE]]

  [ADD [CONSTRAINT [ConstraintName]] tableConstraintDefinition]
  [DROP CONSTRAINT ConstraintName [RESTRICT | CASCADE]]

  [ALTER [COLUMN] SET DEFAULT defaultOption]
  [ALTER [COLUMN] DROP DEFAULT]

- *Drop Table*

  Menghapus tabel berlebihan dari *database* menggunakan pernyataan DROP TABLE, yang memiliki format:

  DROP TABLE TableName [RESTRICT | CASCADE]

- *Create Index*

  Indeks adalah struktur yang menyediakan akses akselerasi ke baris tabel berdasarkan nilai-nilai dari satu atau lebih kolom. Kehadiran indeks dapat secara signifikan meningkatkan kinerja query. Namun, karena indeks dapat diperbarui oleh sistem setiap kali tabel yang mendasari diperbarui, overhead tambahan mungkin timbul. Indeks biasanya dibuat untuk memenuhi kriteria pencarian tertentu setelah tabel telah digunakan selama beberapa waktu dan telah berkembang. Pembuatan indeks bukan standar SQL. Namun, sebagian besar dialek mendukung setidaknya kapabilitasnya sebagai berikut:

  CREATE [UNIQUE] **INDEX** IndexName
  ON TableName (columnName [ASC | DESC] [, . . .])

- *Drop Index*

  Pernyataan DROP INDEX untuk menghapus indeks dari database. DROP INDEX memiliki format:

  DROP INDEX IndexName

2.1.6.2 *Data Manipulation Language* (DML)

Menurut Connolly, T.M., et al. (2010), *data manipulation language* adalah sebuah bahasa yang menyediakan seperangkat operasi untuk mendukung operasi manipulasi data dasar pada data yang terdapat di *database*.Operasi manipulasi data biasanya meliputi:

- Memasukkan data baru ke dalam database
- Modifikasi data yang tersimpan dalam database
- Pengambilan data yang terdapat dalam database

- Penghapusan data dari database

Pernyataan (*statements*) SQL DML meliputi:

- SELECT

  Untuk query data dalam database. Tujuan dari pernyataan SELECT adalah untuk mengambil dan menampilkan data dari satu atau lebih tabel database. Ini adalah perintah sangat handal yang mampu melakukan setara dengan aljabar relasional Seleksi, Proyeksi, dan Gabung operasi dalam sebuah pernyataan tunggal. SELECT adalah perintah yang paling sering digunakan SQL dan memiliki bentuk umum sebagai berikut:

  SELECT [DISTINCT | ALL] {* | [columnExpression [AS newName]] [, . . .]}
  FROM TableName [alias] [, . . .]
  [WHERE condition]
  [GROUP BY columnList]
  [HAVING condition]
  [ORDER BY columnList]

- INSERT

  Ada dua bentuk pernyataan INSERT. Yang pertama memungkinkan satu baris yang akan dimasukkan ke dalam tabel yang bernama dan memiliki format berikut:

  INSERT INTO TableName [(columnList)]
  VALUES (dataValueList)

- UPDATE

  Pernyataan UPDATE memungkinkan isi baris yang ada di tabel yang bernama diubah. Format perintahnya adalah:

  UPDATE TableName
  SET columnName1 = dataValue1 [, columnName2 = dataValue2 . . .]
  [WHERE searchCondition]

- DELETE

  Pernyataan DELETE memungkinkan baris yang akan dihapus dari tabel bernama. Format perintahnya adalah:

  DELETE FROM TableName

  [WHERE searchCondition]

DML dibedakan oleh pengambilan kontruksi yang mendasarinya. yang dibedakan antara dua jenis yaitu:

- *Procedural* DML

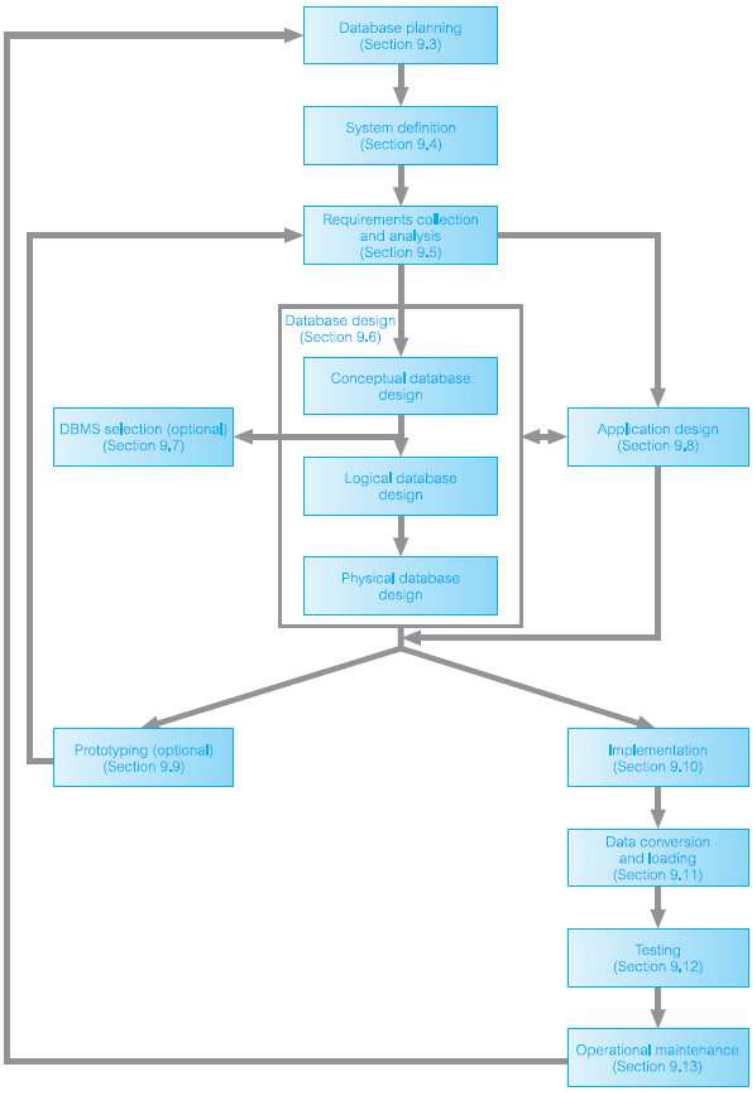
  Sebuah bahasa yang memungkinkan pengguna untuk memberitahu sistem data apa yang dibutuhkan dan bagaimana untuk mengambil data.

- *Nonprocedural* DML

  Sebuah bahasa yang memungkinkan pengguna untuk menyatakan data apa yang dibutuhkan lebih daripada bagaimana data harus diambil.

2.1.7 *Database System Development Lifecycle*

Menurut Connolly, T.M., et al. (2010), *database system development lifecycle* pada dasarnya terkait dengan siklus hidup sistem informasi (SDLC), sebagai sistem database dimana komponen fundamental dari sistem informasi yang lebih besar dari organisasi secara luas.



Gambar 2.2 Tahapan *Database System Development Lifecycle* (Connolly, T.M., et al. (2010))

Berikut tahapan dalam *Database System Development Lifecycle* :

2.1.7.1*Database Planning*

Menurut Connolly, T.M., et al. (2010), *database planning* adalah kegiatan manajemen yang memungkinkan tahapan siklus pengembangan sistem *database* untuk direalisasikan seefisien dan seefektif mungkin.

Perencanaan database harus terintegrasi dengan keseluruhan strategi sistem informasi organisasi. Ada tiga isu utama yang terlibat dalam merumuskan strategi sistem informasi, yaitu:

- Identifikasi rencana perusahaan dan tujuan dengan penentuan berikutnya kebutuhan sistem informasi
- Evaluasi sistem informasi saat ini untuk menentukan kekuatan dan kelemahan yang ada
- Penilaian peluang IT yang mungkin menghasilkan keunggulan kompetitif

Langkah pertama yang penting dalam perencanaan database dengan jelas mendefinisikan *mision statement* untuk sistem database. *Mision statement* mendefinisikan tujuan utama dari sistem database. Mereka mendorong proyek database dalam organisasi (seperti Direktur dan / atau pemilik) biasanya menentukan *mision statement*. Sebuah *mision statement* membantu untuk memperjelas tujuan dari sistem database dan memberikan jalan yang lebih jelas menuju terciptanya efisien dan efektif sistem database yang diperlukan.

Setelah *mision statement* didefinisikan, aktivitas berikutnya melibatkan mengidentifikasi *mission objectives.* Setiap *mission objectives* harus mengidentifikasi tugas tertentu dimana sistem database harus mendukung. Asumsinya adalah bahwa jika sistem database mendukung *mission objectives* maka *mission statement* harus dipenuhi.

Mission statement dan *mission objectives* bisa disertai dengan beberapa informasi tambahan yang menentukan, secara umum, pekerjaan yang harus dilakukan, sumber daya yang dapat digunakan untuk melakukannya, dan uang untuk membayar untuk itu semua.

2.1.7.2 *System Definition*

Menurut Connolly, T.M., et al. (2010), *system definition* adalah menjelaskan ruang lingkup dan batas-batas dari aplikasi *database* dan pandangan pengguna utama.

2.1.7.3 *Requirements Collection and Analysis*

Menurut Connolly, T.M., et al. (2010), *requirements collection and analysis* adalah proses mengumpulkan dan menganalisis informasi tentang bagian dari organisasi yang akan didukung oleh sistem *database*, dan menggunakan informasi ini untuk mengidentifikasi persyaratan untuk sistem baru.

Tahap ini melibatkan pengumpulan dan analisis informasi tentang bagian dari perusahaan yang akan dilayani oleh *database*. Ada banyak teknik untuk mengumpulkan informasi ini, yang disebut fact-finding techniques (teknik pencarian fakta). Informasi dikumpulkan untuk setiap *user view* utama (yaitu, *job role* atau area aplikasi *enterprise*), termasuk:

- Deskripsi data yang digunakan atau dihasilkan
- Rincian tentang bagaimana data akan digunakan atau dihasilkan
- Persyaratan tambahan untuk sistem *database* baru

Ada tiga pendekatan utama untuk mengelola persyaratan sistem *database* dengan *multiple user views*, yaitu:

- *Centralized Approach*

  Persyaratan untuk setiap tampilan pengguna yang bergabung ke satu set persyaratan untuk sistem *database* baru. Sebuah model data yang mewakili semua *user views* dibuat selama tahap desain *database*.
- *View Integration Approach*

  Persyaratan untuk setiap *user views* tetap sebagai daftar terpisah. Model data yang mewakili setiap *user views* diciptakan dan kemudian bergabung selama tahap desain *database*.
- Kombinasi kedua pendekatan

  Menggunakan pencekatan *centralized* dan *view integration.*

2.1.7.4*Database Design*

Menurut Connolly, T.M., et al. (2010), *database design* adalah proses menciptakan desain yang akan mendukung rumusan misi dan tujuan misi perusahaan untuk sistem basis data yang diperlukan.

Beberapa pendekatan dalam *database design* (perancangan basis data):

- *Bottom Up*

  Pendekatan *bottom-up* dimulai pada tingkat dasar atribut (yaitu, properti entitas dan relasi), yang melalui analisis relasi antara atribut, dikelompokkan ke dalam relasi yang mewakili jenis entitas dan relasi antar entitas.
- *Top Down*

  Sebuah strategi yang lebih tepat untuk desain database yang kompleks. Pendekatan ini dimulai dengan pengembangan model data yang berisi beberapa entitas dan relasi tingkat tinggi dan kemudian menerapkan

perbaikan *top-down* berurutan untuk mengidentifikasi entitas dengan level lebih rendah, relasi, dan atribut terkait. Pendekatan top-down diilustrasikan menggunakan konsep *Entity-Relationship* (ER) model, dimulai dengan identifikasi entitas dan relasi antara entitas yang menarik bagi organisasi.

- *Inside Out*

  Dalam pendekatan *inside-out* ini terkait dengan pendekatan *bottom-up* tetapi berbeda dengan terlebih dahulu mengidentifikasi satu set entitas utama dan kemudian menyebar keluar untuk mempertimbangkan entitas lain, relasi, dan atribut yang terkait dengannnya pertama kali diidentifikasi.

- *Mixed Strategy*

  Pendekatan strategi campuran menggunakan kedua pendekatan *bottom-up* dan *top-down* untuk berbagai bagian dari model sebelum akhirnya menggabungkan semua bagian bersama-sama.

Database desain terdiri dari tiga tahap utama, yaitu *conceptual, logical,* dan *physical.*

- *Conceptual Database Design*

  Proses membangun sebuah model dari data yang digunakan dalam suatu perusahaan, independen dari semua pertimbangan fisik.

  Pengertian *conceptual database design* menurut Hongji Zhang, Xuping Li, Yong Luo, Lianze Teng, dan Aiqun Dai (2013, 121) *conceptual database design* adalah abstrak data ke dalam model konseptual dipahami oleh pengguna.

- *Logical Database Design*

  Proses membangun sebuah model dari data yang digunakan dalam suatu perusahaan berdasarkan pada

model data yang spesifik, tetapi independen dari DBMS tertentu dan pertimbangan fisik lainnya.

- *Physical Database Design*

  Proses menghasilkan penjelasan implementasi *database* pada *secondary storage*; menggambarkan hubungan dasar, file organisasi, indeks yang digunakan untuk mencapai akses yang efisien terhadap data, dan setiap *integrity constraints* terkait dan langkah-langkah keamanan.

2.1.7.5 DBMS *Selection* (*Optional*)

Menurut Connolly, T.M., et al. (2010), DBMS *Selection* adalah proses pemilihan suatu DBMS yang tepat untuk mendukung sistem *database*.

Langkah-langkah utama untuk memilih DBMS:

- Tentukan kerangka acuan studi

  Kerangka acuan untuk pemilihan DBMS dibentuk, menyatakan tujuan dan ruang lingkup penelitian, dan tugas-tugas yang perlu dilakukan. Dokumen ini juga dapat mencakup deskripsi kriteria (berdasarkan spesifikasi kebutuhan pengguna) yang akan digunakan untuk mengevaluasi produk DBMS, daftar awal kemungkinan produk, dan semua kendala yang diperlukan dan rentang waktu untuk penelitian.

- Daftar *shortlist* dua atau tiga produk

  Kriteria dianggap 'kritis' untuk keberhasilan pelaksanaan dapat digunakan untuk menghasilkan daftar awal produk DBMS untuk evaluasi.

- Mengevaluasi produk

  Ada berbagai fitur yang dapat digunakan untuk mengevaluasi produk DBMS. Untuk keperluan evaluasi, fitur ini dapat dinilai sebagai kelompok (misalnya,

definisi data) atau secara individu (misalnya, jenis data yang tersedia).

- Merekomendasikan pilihan dan menghasilkan laporan

  Langkah terakhir dari pemilihan DBMS adalah untuk mendokumentasikan proses dan untuk memberikan pernyataan temuan dan rekomendasi untuk produk DBMS tertentu.

2.1.7.6*Application Design*

Menurut Connolly, T.M., et al. (2010), *application design* adalah desain *user interface* dan program aplikasi yang menggunakan dan memproses *database*.

Dua aspek *application design*, yaitu *transaction design* dan *user interface design*.

- *Transaction Design*

  Transaksi sebagai tindakan, atau serangkaian tindakan yang dilakukan oleh pengguna tunggal atau program aplikasi, yang mengakses atau mengubah isi *database*.

  Sebuah transaksi dapat terdiri dari beberapa operasi, seperti transfer uang dari satu *account* ke *account* lainnya.

  Tujuan dari desain transaksi adalah untuk menetapkan dan mendokumentasikan karakteristik tingkat tinggi dari transaksi yang dibutuhkan pada *database*, termasuk:

  - Data yang akan digunakan oleh transaksi;
  - Karakteristik fungsional dari transaksi;
  - Output transaksi;
  - Kepentingan bagi pengguna;
  - Tingkat pengembalian yang diharapkan dari penggunaan.

Ada tiga jenis utama transaksi:

- Retrieval transactions

  Digunakan untuk mengambil data untuk ditampilkan di layar atau dalam produksi laporan. Sebagai contoh, operasi untuk mencari dan menampilkan rincian dari properti (diberi nomor properti) adalah contoh dari transaksi pengambilan.

- Update transactions

  Digunakan untuk menyisipkan catatan baru, menghapus catatan lama, atau memodifikasi catatan yang ada dalam *database*. Sebagai contoh, operasi untuk memasukkan rincian properti baru ke dalam *database* adalah contoh dari sebuah transaksi *update.*

- *Mixed transactions*

  Melibatkan baik pengambilan dan memperbarui data. Sebagai contoh, operasi untuk mencari dan menampilkan rincian dari properti (diberi nomor properti) dan kemudian memperbarui nilai sewa bulanan adalah contoh dari transaksi campuran.

- *User Interface Design*

  Sebelum menerapkan suatu bentuk atau laporan, adalah penting bahwa pertama-tama merancang tata letak. Berikut panduan yang berguna (*User Interface Design Guidelines*) untuk diikuti saat merancang bentuk atau laporan:

  - *Meaningful title*

    Informasi yang disampaikan oleh judul harus jelas dan tegas mengidentifikasi tujuan form/laporan.

  - *Comprehensible instructions*

    Istilah familiar harus digunakan untuk menyampaikan instruksi kepada pengguna. Petunjuk harus singkat, dan, ketika lebih banyak informasi

yang diperlukan, bantuan layar harus tersedia. Instruksi harus ditulis dalam gaya tata bahasa yang konsisten menggunakan format standar.

- *Logical grouping and sequencing of fields*

  Bidang terkait harus diposisikan bersama-sama pada form/laporan. Urutan bidang harus logis dan konsisten.

- *Visually appealing layout of the form/report*

  Form/laporan harus menyajikan antarmuka yang menarik bagi pengguna. Form/laporan akan muncul seimbang dengan field atau kelompok field secara merata ditempatkan di seluruh form/laporan. Tidak boleh ada area dari form/laporan yang memiliki terlalu sedikit atau terlalu banyak field. Field atau kelompok field harus dipisahkan oleh sejumlah ruang reguler. Apabila diperlukan, field harus secara vertikal atau horizontal. Dalam kasus di mana form di layar sama dengan *hardcopy*, penampilan keduanya harus konsisten.

- *Familiar field labels*

  Label field harus familiar. Sebagai contoh, jika '*Sex'* digantikan oleh '*Gender'*, kemungkinan beberapa pengguna akan bingung.

- *Consistent terminology and abbreviations*

  Sebuah daftar yang telah disetujui atau istilah yang familiar dan singkatan harus digunakan secara konsisten.

- *Consistent use of color*

  Warna harus digunakan untuk memperbaiki penampilan form/laporan dan untuk menyoroti field penting atau pesan penting. Untuk mencapai hal ini, warna harus digunakan dengan cara yang konsisten dan bermakna. Misalnya, bidang pada formulir dengan latar belakang putih dapat mengindikasikan

field data-entry dan orang-orang dengan latar belakang biru dapat menunjukkan field *display-only*.

- *Visible space and boundaries for data-entry fields*

  Seorang pengguna harus secara visual menyadari jumlah total ruang yang tersedia untuk masing-masing field. Hal ini memungkinkan pengguna untuk mempertimbangkan format yang sesuai untuk data sebelum memasuki nilai-nilai ke dalam field.

- *Convenient cursor movement*

  Seorang pengguna harus mudah mengidentifikasi operasi yang dibutuhkan untuk memindahkan kursor seluruh form/laporan. Mekanisme sederhana seperti menggunakan tombol *tab*, panah, atau *pointer mouse* harus digunakan.

- *Error correction for individual characters and entire fields*

  Seorang pengguna harus mudah mengidentifikasi operasi yang dibutuhkan untuk membuat perubahan untuk nilai field. Mekanisme sederhana harus tersedia seperti menggunakan tombol *backspace* atau dengan *overtyping*.

- *Error messages for unacceptable values*

  Jika pengguna mencoba untuk memasukkan data yang salah ke dalam field, pesan kesalahan akan ditampilkan. Pesan harus menginformasikan pengguna dari kesalahan dan menunjukkan nilai-nilai yang diijinkan.

- *Optional fields marked clearly*

  Bidang opsional harus secara jelas diidentifikasi bagi pengguna. Hal ini dapat dicapai dengan menggunakan label field sesuai atau dengan menampilkan field menggunakan warna yang menunjukkan jenis field. Field opsional harus ditempatkan setelah field wajib.

- *Explanatory messages for fields*

  Ketika seorang pengguna menempatkan kursor di field, informasi tentang field akan muncul dalam posisi reguler di layar seperti *window status bar*.

- *Completion signal*

  Ini harus jelas bagi pengguna ketika proses mengisi kolom pada formulir selesai. Namun, pilihan untuk menyelesaikan proses tidak harus otomatis sebagai pengguna sebaiknya meninjau data yang dimasukkan.

2.1.7.7 *Prototyping* (*Optional*)

Menurut Connolly, T.M., et al. (2010), *prototyping* adalah proses membangun model kerja dari sistem *database*.

Sebuah prototipe adalah model kerja yang biasanya tidak memiliki semua fitur yang diperlukan atau menyediakan semua fungsi sistem final. Tujuan utama dari pengembangan sistem *database* prototipe adalah untuk memungkinkan pengguna untuk menggunakan prototipe untuk mengidentifikasi fitur dari sistem yang bekerja dengan baik, atau tidak memadai, dan jika mungkin untuk menyarankan perbaikan atau fitur baru bahkan ke sistem *database*.

Ada dua strategi prototyping umum digunakan saat ini:

- *Requirements Prototyping*

  Menggunakan prototipe untuk menentukan persyaratan sistem database yang diusulkan dan setelah persyaratan lengkap prototipe tersebut akan dibuang.

- Evolutionary Prototyping

  Digunakan untuk tujuan yang sama, perbedaan penting adalah bahwa prototipe tidak dibuang tetapi dengan pengembangan lebih lanjut menjadi sistem *database* yang bekerja.

2.1.7.8 *Implementation*

Menurut Connolly, T.M., et al. (2010), *implementation* adalah proses realisasi fisik dari *database* dan aplikasi desain.

Implementasi database dicapai dengan menggunakan Data Definition Language (DDL) dari DBMS yang dipilih atau Graphical User Interface (GUI), yang menyediakan fungsi yang sama sambil menyembunyikan laporan DDL tingkat rendah. Laporan DDL digunakan untuk membuat struktur database dan file database kosong. Setiap *user view* tertentu juga diimplementasikan pada tahap ini.

2.1.7.9 *Data Conversion and Loading*

Menurut Connolly, T.M., et al. (2010), *data conversion and loading* adalah proses mentransfer data yang ada ke dalam *database* baru dan mengkonversi aplikasi yang ada untuk dijalankan pada *database* baru.

Tahap ini diperlukan hanya ketika sistem *database* baru menggantikan sistem yang lama. Saat ini adalah umum untuk DBMS untuk memiliki utilitas yang memuat file yang sudah ada ke dalam *database* baru. Utilitas biasanya membutuhkan spesifikasi sumber file dan target *database*, dan kemudian secara otomatis mengkonversi data ke format yang diperlukan dari file *database* baru.

Mana yang berlaku, dimungkinkan bagi pengembang untuk mengkonversi dan menggunakan program aplikasi dari sistem lama untuk digunakan oleh sistem baru. Setiap kali konversi dan pemuatan diperlukan, proses tersebut harus direncanakan dengan baik untuk memastikan kelancaran transisi ke operasi penuh.

2.1.7.10 *Testing*

Menurut Connolly, T.M., et al. (2010), *testing* adalah proses menjalankan sistem *database* dengan maksud menemukan kesalahan.

Berikut kriteria dan contoh yang dapat digunakan untuk melakukan evaluasi:

- *Learnability* - Berapa lama waktu yang dibutuhkan pengguna baru untuk menjadi produktif dengan sistem?
- *Performance* - Seberapa baik respon sistem sesuai praktek kerja pengguna?
- *Robustness* - Seberapa toleran sistem kesalahan pengguna?
- *Recoverability* - Seberapa baik sistem dipulihkan dari kesalahan pengguna?
- *Adapatability* - Seberapa dekat sistem terikat satu model kerja?

2.1.7.11 *Operational Maintenance*

Menurut Connolly, T.M., et al. (2010), *operational maintenance* adalah proses pemantauan dan pemeliharaan sistem *database* berikut instalasi.

Kegiatan-kegiatan dalam tahapan *maintenance* (pemeliharaan):

- Pemantauan kinerja sistem.

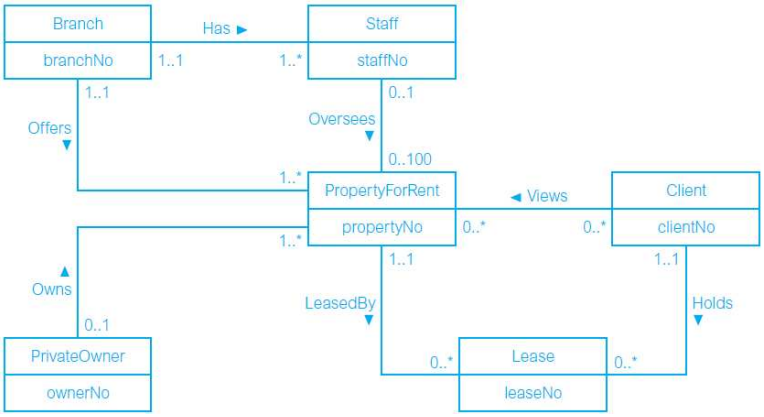
  Jika kinerja turun di bawah tingkat yang dapat diterima, penyetelan atau reorganisasi *database* mungkin diperlukan.
- Mempertahankan dan meningkatkan sistem *database* (bila diperlukan).

  Persyaratan baru yang dimasukkan ke dalam sistem *database* melalui tahapan sebelumnya dari siklus hidup.

2.1.8 *Entity Relationship Diagram / Modeling*

Menurut Connolly, T.M., et al. (2010), *entity relationship diagram / modeling* adalah pendekatan *top-down* untuk desain *database* yang dimulai dengan mengidentifikasi data penting yang disebut entitas dan relasi antara data yang harus direpresentasikan dalam model.

Kemudian menambahkan rincian seperti informasi yang ingin mempertahankan entitas dan relasi yang disebut atribut dan setiap kendala pada entitas, relasi, dan atribut. Pemodelan ER adalah teknik penting untuk setiap desainer database untuk menguasai dan membentuk dasar dari metodologi.

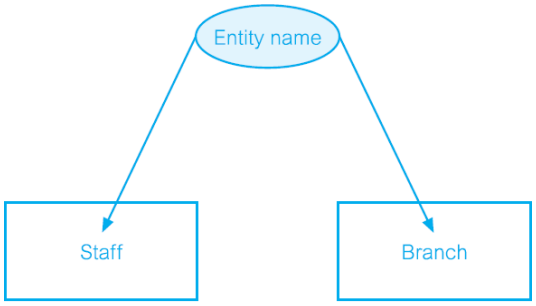


Gambar 2.3 Contoh *Entity Reltionship Diagram*
(Connolly, T.M., et al. (2010))

2.1.8.1 *Entity Types*

Menurut Connolly, T.M., et al. (2010), *entity types* adalah sekelompok objek dengan properti yang sama, yang diidentifikasi oleh perusahaan dengan memiliki eksistensi independen.

Entity occurrence sebagai sebuah objek yang dapat diidentifikasi secara unik dari suatu entitas.



Gambar 2.4 Contoh *Entity Types* 'Staff' dan 'Branch' (Connolly, T.M., et al. (2010))

Konsep dasar dari model ER adalah *entity types*, yang mewakili sekelompok 'benda' di 'dunia nyata' dengan sifat yang sama. Suatu *entity types* memiliki eksistensi independen dan dapat menjadi objek dengan eksistensi *physical* (nyata) atau objek dengan eksistensi *conceptual* (abstrak), seperti yang tercantum pada Gambar 2.4.

Physical existence	
Staff	Part
Property	Supplier
Customer	Product
Conceptual existence	
Viewing	Sale
Inspection	Work experience

Gambar 2.5 Contoh Entitas Dengan Eksistensi *Physical* atau *Conceptual* (Connolly, T.M., et al. (2010))

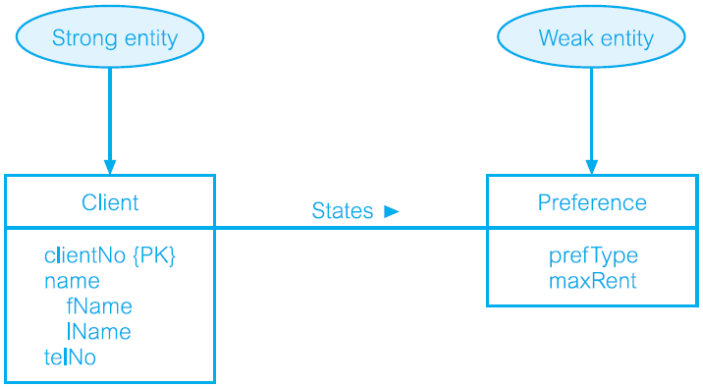
Klasifikasi *entity types* menjadi *strong* dan *weak*:

- *Strong Entity Type*

  Suatu entitas yang keberadaanya tidak tergantung pada entitas lain. *Strong Entity Type* disebut sebagai *parent, owner*, atau entitas dominan.

- *Weak Entity Type*

  Suatu entitas keberadaannya tergantung pada entitas lain. *Weak entity type* kadang-kadang disebut sebagai *child, dependent,* atau entitas *subordinate*.

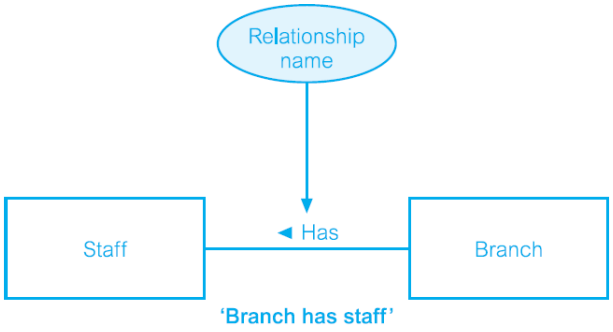


Gambar 2.6 Contoh *Strong Entity* dan *Weak Entity* (Connolly, T.M., et al. (2010))

2.1.8.2 *Relationship Types*

Menurut Connolly, T.M., et al. (2010), *relationship types* adalah satu set asosiasi yang mempunyai arti antara jenis entitas.

Relationship occurrence adalah sebuah asosiasi yang dapat diidentifikasi secara unik, yang meliputi satu kejadian dari setiap tipe entitas yang berpartisipasi.



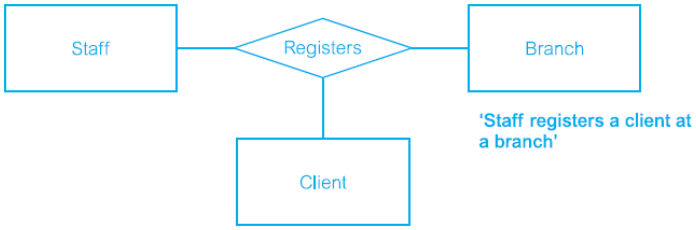
Gambar 2.7 Contoh *Relationship Types* 'Branch Has Staff' (Connolly, T.M., et al. (2010))

Degree of relationship type adalah jumlah partisipasi jenis entitas dalam sebuah relasi. Entitas yang terlibat dalam jenis relasi tertentu yang disebut sebagai *participants* dalam relasi itu. Jumlah *participants* dalam jenis relasi yang disebut *degree* dari relasi itu. Oleh karena itu, tingkat relasi menunjukkan jumlah jenis entitas yang terlibat dalam suatu relasi. *Degree* dari relasi berupa *binary*, *ternary* dan *quarternary*.

Binary sebagai relasi dengan dua *degree*. Pada kenyataannya tingkat yang paling umum untuk relasi adalah *binary* seperti yang ditunjukkan dalam gambar ini.

Gambar 2.8 Contoh Relasi *Binary* 'PrivateOwner memiliki PropertyForRent' (Connolly, T.M., et al. (2010))

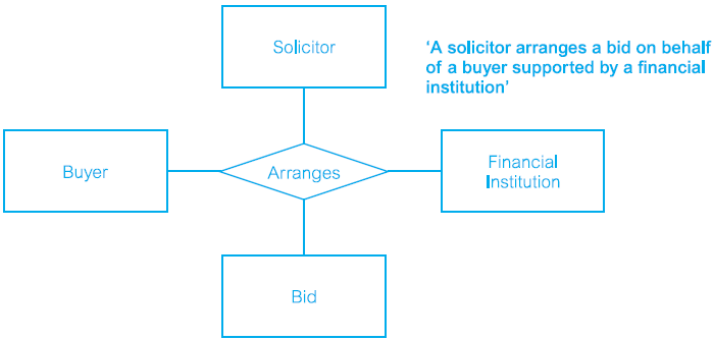
Ternary sebagai relasi dengan tiga *degree*. Berikut contoh relasi *ternary*:



Gambar 2.9 Contoh Relasi *Ternary* 'Registers' (Connolly, T.M., et al. (2010))

Contoh menunjukkan relasi register dimana terdapat tiga jenis entitas yang berpartisipasi, yaitu Staff, Branch, dan Client. Relasi ini merupakan pendaftaran Client oleh seorang anggota Staff di Branch. Istilah 'relasi yang kompleks' digunakan untuk menjelaskan relasi dengan derajat (*degree*) lebih tinggi dari *binary*.

Quarternary sebagai relasi dengan empat *degree*. Berikut contoh relasi *quarternary*:

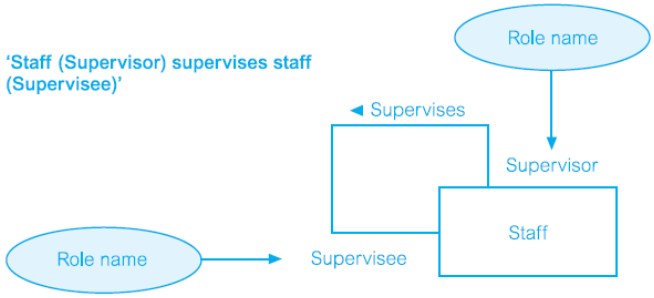


Gambar 2.10 Contoh Relasi *Quarternary* 'Arranges' (Connolly, T.M., et al. (2010))

Relasi 'Arranges' dengan empat tipe entitas yang berpartisipasi, yaitu Buyer, Solicitor, FinancialInstitution, dan Bid pada Gambar 2.8. Relasi ini merupakan situasi di mana pembeli, disarankan oleh pengacara dan didukung oleh lembaga keuangan, menempatkan tawaran.

Recursive Relationship adalah jenis relasi di mana jenis entitas yang sama berpartisipasi lebih dari sekali dalam peran yang berbeda. Relasi rekursif kadang-kadang disebut juga dengan relasi *unrary*.

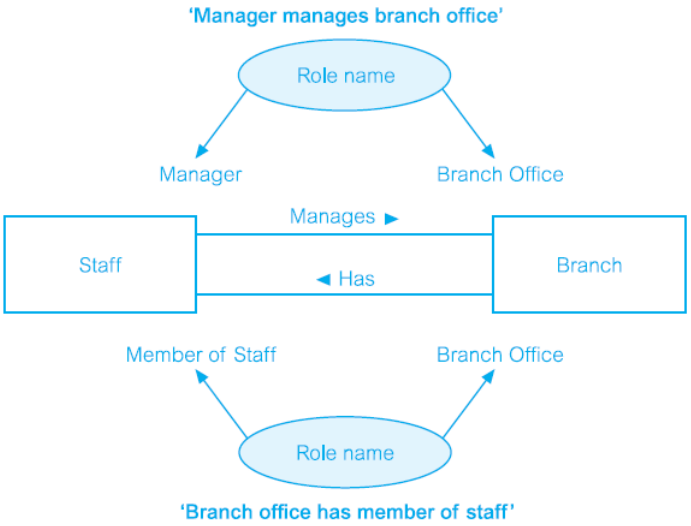
Relasi dapat diberikan sebuah *role names* untuk menunjukkan tujuan bahwa setiap jenis entitas yang berpartisipasi bermain dalam suatu hubungan. *Role names* dapat menjadi penting bagi relasi rekursif untuk menentukan fungsi dari masing-masing peserta.



Gambar 2.11 Contoh *Recursive Relationship* 'Supervises' dengan Role Name Supervisor dan Supervisee (Connolly, T.M., et al. (2010))

Relasi rekursif diatas disebut 'Supervises', yang merupakan asosiasi Staff dengan Supervisor dimana Supervisor juga anggota Staff. Dengan kata lain, tipe entitas Staff berpartisipasi dua kali. Partisipasi pertama dari entitas Staf f dalam relasi 'Supervises' diberi nama peran 'Supervisor' dan partisipasi kedua diberi nama peran 'Supervisee' sebagai anggota Staff yang diawasi.

Role names juga dapat digunakan ketika dua entitas yang terkait melalui lebih dari satu hubungan.



Gambar 2.12 Contoh *Recursive Relationship* 'Manages dan Has' dengan Role Name Manager, Member of Staff, dan Branch Office (Connolly, T.M., et al. (2010))

Entitas Staff dan Branch berhubungan melalui dua relasi yang berbeda yang disebut Manages dan Has. Staf f Manages Branch, Member of Staff diberi nama peran ‘Manager’ mengelola Branch diberi nama peran ‘Branch Office’. Demikian pula, untuk Branch Has Staff, sebuah Branch, diberi nama peran ‘Branch Office’ memiliki Staff diberi nama peran ‘Member of Staff’.

2.1.8.3*Attributes*

Menurut Connolly, T.M., et al. (2010), *attributes* adalah sebuah properti dari suatu entitas atau tipe relasi. Atribut menyimpan nilai yang menggambarkan setiap kejadian entitas dan mewakili bagian utama dari data yang disimpan dalam *database*. Contoh atribut Staff memiliki atribut staffNo, name, position, dan salary.

Attribute domain adalah kumpulan nilai yang diijinkan untuk satu atau lebih atribut. Domain mendefinisikan nilai-nilai potensial dari atribut yang disimpan dan mirip dengan konsep domain dalam model relasional. Contoh domain untuk atribut alamat terdiri dari subdomain: jalan, kota dan kode pos.

Domain dari nama atribut lebih sulit untuk didefinisikan, karena hanya terdiri dari semua nama yang mungkin. Hal ini tentunya karakter string, tapi mungkin terdiri tidak hanya dari huruf tetapi juga dari tanda hubung atau karakter khusus lainnya. Sebuah model data sepenuhnya dikembangkan meliputi domain dari setiap atribut dalam model ER.

Dalam atribut dapat diklasifikasikan dalam beberapa jenis yaitu:

- *Simple and Composite Attributes*

  Simple Attributes adalah sebuah atribut yang terdiri dari komponen tunggal dengan eksistensi independen. Atribut

sederhana tidak dapat dibagi lagi menjadi komponen yang lebih kecil. Contoh atribut sederhana meliputi Position dan Salary dari entitas Staff. Atribut sederhana kadang-kadang disebut atribut atom.

Composite Attributes adalah sebuah atribut yang terdiri dari beberapa komponen, masing-masing dengan eksistensi independen. Beberapa atribut selanjutnya dapat terbagi untuk menghasilkan komponen yang lebih kecil dengan eksistensi independen mereka sendiri. Contoh atribut alamat terbagi menjadi jalan, kota, dan kode pos.

- *Single Valued and Multi Valued Attributes*

  Single Valued Attributes adalah sebuah atribut yang memegang nilai tunggal untuk setiap kejadian dari suatu entitas. Mayoritas dari atribut bernilai tunggal. Sebagai contoh, setiap kemunculan jenis entitas Branch memiliki nilai tunggal untuk atribut branchNo (misalnya B003), dan karena itu atribut branchNo disebut bernilai tunggal.

  Multi Valued Attributes adalah sebuah atribut yang memegang beberapa nilai untuk setiap kejadian dari suatu entitas. Sebagai contoh, setiap kemunculan entitas Branch dapat memiliki beberapa nilai untuk atribut telNo (misalnya, nomor Branch B003 memiliki nomor telepon 0141-339-2178 dan 0141-339-4439) dan oleh karena itu atribut telNo dalam hal ini multi -valued. Dengan kata lain, Branch mungkin memiliki minimal satu nomor telepon untuk maksimal tiga nomor telepon

- *Derived Attributes*

  Derived Attributes adalah sebuah atribut yang mewakili nilai yang diturunkan dari nilai atribut terkait atau set atribut, belum tentu dalam jenis entitas yang sama.

Sebagai contoh, nilai untuk atribut durasi dari entitas Lease dihitung dari atribut RentStart dan rentFinish. Jadi, atribut durasi sebagai atribut turunan, dimana nilainya berasal dari RentStart dan rentFinish atribut.

Dalam atribut terdapat beberapa jenis atribut yang tergabung dalam keys yaitu:

- *Super Key*

  Super Key adalah atribut atau kumpulan atribut yang secara unik mengidentifikasi sebuah tuple dalam relasi. Namun, *super key* mungkin berisi atribut tambahan yang tidak diperlukan untuk identifikasi yang unik, dan superkeys yang hanya berisi jumlah minimum atribut yang diperlukan untuk identifikasi yang unik.

- *Candidate Key*

  Candidate Key adalah seperangkat minimal atribut yang secara unik mengidentifikasi setiap kemunculan suatu entitas.

  Sebagai contoh, atribut branchNo adalah *candidate key* untuk entitas Branch, dan memiliki nilai yang berbeda untuk setiap kejadian entitas Branch. *Candidate key* harus menyimpan nilai yang unik untuk setiap kejadian dari suatu entitas. Ini berarti bahwa *candidate key* tidak dapat berisi *null*.

- *Primary Key*

  Primary Key adalah *candidate key* yang dipilih untuk secara unik mengidentifikasi setiap kemunculan suatu entitas. Pilihan *primary key* untuk suatu entitas didasarkan pada pertimbangan panjang atribut, jumlah minimal atribut yang diperlukan, dan kepastian masa depan keunikan.

Misalnya, staffNo perusahaan didefinisikan mengandung maksimal lima karakter (misalnya, SG14) sedangkan NIN (nomor induk asuransi) berisi maksimal sembilan karakter (misalnya, WL220658D). Oleh karena itu, kita pilih staffNo sebagai primary key dari entitas Staff dan NIN kemudian disebut sebagai *alternate key*.

- *Composite Key*

  Composite Key adalah sebuah *candidate key* yang terdiri dari dua atau lebih atribut.

  Sebagai contoh, sebuah entitas yang disebut Advert dengan atribut propertyNo, newspaperName, dateAdvert, dan biaya. Banyak properti yang diiklankan di banyak surat kabar pada tanggal tertentu. Untuk mengidentifikasi unik setiap kemunculan entitas Advert membutuhkan nilai-nilai untuk propertyNo, newspaperName, dan dateAdvert atribut. Dengan demikian, entitas Advert memiliki kunci primer komposit terdiri dari atribut propertyNo, newspaperName, dan dateAdvert.

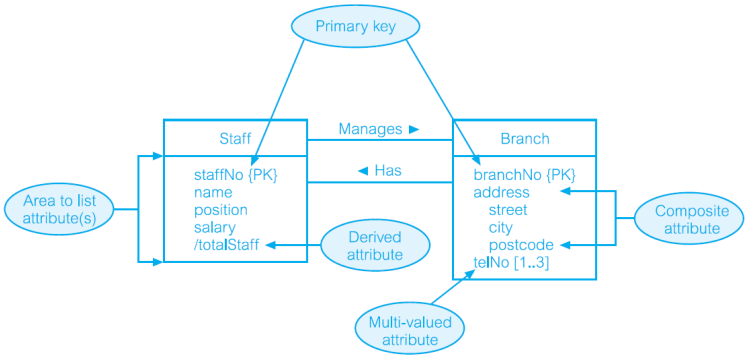
- *Foreign Key*

  Foreign Key adalah atribut atau kumpulan atribut dalam satu relasi yang cocok dengan candidate key dari beberapa kemungkinan relasi yang sama.

  Contoh, entitas Staff dan Branch memiliki relasi dimana pada Staff memiliki atribute PK dari Branch yaitu branchNo, sehingga branchNo menjadi FK pada entitas Staff.

- *Alternate Key*

  Alternate Key adalah *candidate key* yang tidak dipilih menjadi *primary key*.



Gambar 2.13 Representasi Entitas Staff dan Branch dengan Atributnya (Connolly, T.M., et al. (2010))

2.1.8.4 *Structural Constraints*

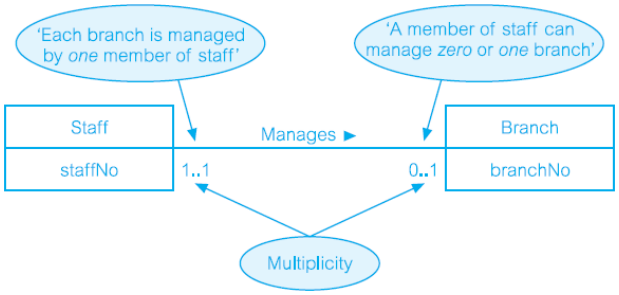
Constraints (kendala) dapat ditempatkan pada entitas yang berpartisipasi dalam suatu relasi. Kendala harus mencerminkan pembatasan pada relasi sebagaimana yang dipersepsikan di 'dunia nyata'. Contoh kendala tersebut meliputi persyaratan bahwa properti untuk disewakan harus memiliki pemilik dan setiap cabang harus memiliki Staff. Jenis utama dari kendala pada relasi disebut *Multiplicity*.

Menurut Connolly, T.M., et al. (2010), *multiplicity* adalah jumlah (atau range) kemungkinan kejadian suatu entitas yang mungkin berhubungan dengan kejadian tunggal dari suatu entitas yang terkait melalui relasi tertentu.

Berikut tiga jenis relasi *binary* yang merupakan *degree* paling umum dalam relasi:

- *One to One* (1:1) *Relationships*

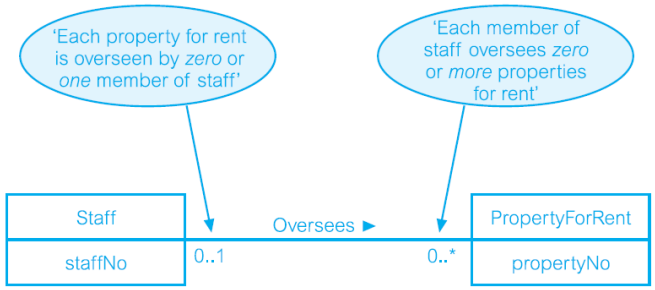
  Relasi *one to one* menampilkan hubungan antara entitas dimana satu entitas menempatkan hanya satu anggota entitas lainnya. Berikut contoh menampilkan bahwa anggota staff dapat mengelola nol atau satu cabang dengan menempatkan 0..1 di samping entitas Branch. Untuk menyatakan bahwa cabang selalu memiliki satu manajer dengan menempatkan sebuah 1..1 di samping entitas Staff.



Gambar 2.14 Contoh *Multiplicity* Untuk *One to One Relationship* (Connolly, T.M., et al. (2010))

- *One to Many* (1:*) *Relationships*

  Relasi *one to many* menampilkan hubungan antara entitas dimana satu entitas dapat menempatkan banyak anggota entitas lain. Berikut contoh menampilkan bahwa anggota staff dapat mengawasi nol atau lebih properti untuk disewakan dengan menempatkan sebuah 0..* 'di samping entitas PropertyForRent. Untuk menyatakan bahwa masing-masing properti untuk disewakan diawasi oleh nol atau satu anggota staff dengan menempatkan sebuah 0..1 di samping entitas Staff.

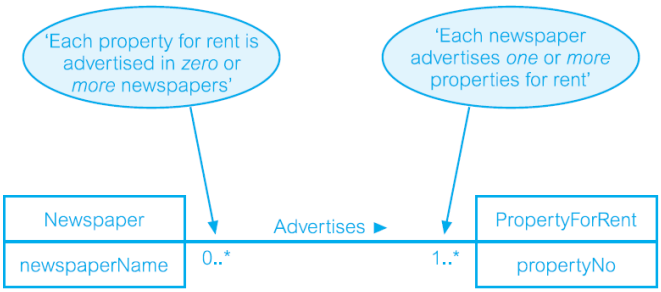


Gambar 2.15 Contoh *Multiplicity* Untuk *One to Many Relationship* (Connolly, T.M., et al. (2010))

- *Many to Many* (*:*) *Relationships*

  Relasi *many to many* menampilkan hubungan antara entitas dimana suatu entitas dapat memiliki lebih dari dari satu anggota entitas lain dan anggota entitas lain

dapat memiliki lebih dari satu entitas. Berikut contoh menampilkan bahwa setiap surat kabar dapat mengiklankan satu atau lebih properti untuk disewakan dengan menempatkan 1..* di samping entitas PropertyForRent. Untuk menyatakan bahwa masing-masing properti untuk disewakan dapat diiklankan oleh nol atau lebih surat kabar dengan menempatkan sebuah 0..* di samping entitas koran.



Gambar 2.16 Contoh *Multiplicity* Untuk *Many to Many Relationship* (Connolly, T.M., et al. (2010))

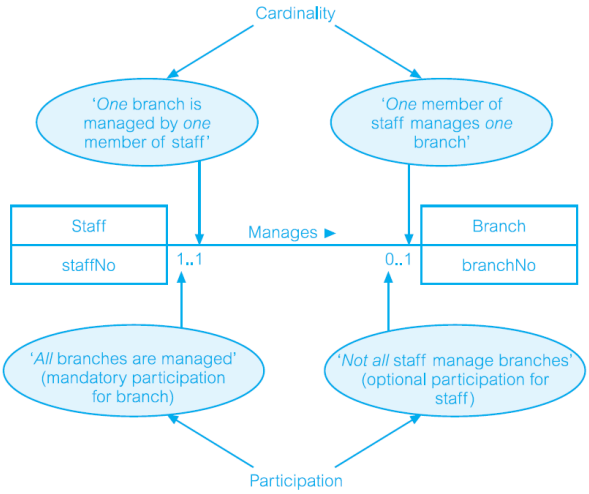
Multiplicity sebenarnya terdiri dari dua *constraints* yang terpisah yang dikenal sebagai:

- *Cardinality*

  Menjelaskan jumlah maksimum kemungkinan relasi yang terjadi untuk entitas yang berpartisipasi dalam relasi yang diberikan.

- *Participation*

  Menentukan apakah semua atau hanya beberapa kejadian entitas berpartisipasi dalam suatu relasi.



Gambar 2.17 Contoh *Multiplicity* Untuk *Cardinality* dan *Participation Constraints* (Connolly, T.M., et al. (2010))

Relasi Manages yang ditunjukkan pada Gambar memiliki cardinality one to one dan ini diwakili oleh rentang banyaknya dengan nilai maksimum 1 di kedua sisi relasi.

Optional Participation untuk entitas Staff pada relasi Manages ditampilkan dengan nilai minimum 0 untuk *multiplicity* samping entitas Branch dan *Mandatory Participation* bagi entitas Cabang di relasi Manages ditampilkan dengam nilai minimum 1 untuk *multiplicity* samping entitas Staff.

2.1.8.5 *Problems with ER Models*

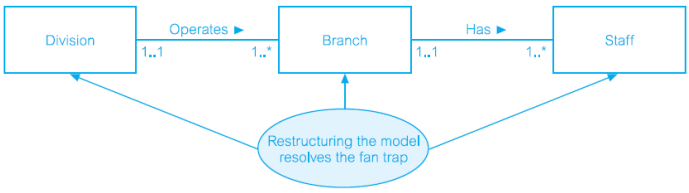
Masalah yang mungkin timbul saat membuat model ER. Masalah-masalah ini disebut sebagai *connection traps*, dan biasanya terjadi karena salah tafsir tentang makna relasi tertentu. Berikut dua jenis *connection traps*:

- *Fan Traps*

  Dimana model merupakan relasi antara tipe entitas, tetapi jalur antara kejadian entitas tertentu adalah ambigu.

Gambar 2.18 Contoh *Fan Traps* (Connolly, T.M., et al. (2010))

Model ini menampilkan fakta-fakta satu divisi beroperasi satu atau lebih cabang dan memiliki satu atau lebih staff. Namun, masalah muncul ketika kita ingin tahu mana anggota staff bekerja di cabang tertentu.

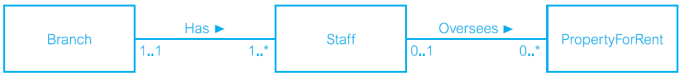


Gambar 2.19 Contoh Restrukturisasi Untuk Menghapus *Fan Traps* (Connolly, T.M., et al. (2010))

Mengatasi *fan traps* ini dengan melakukan restrukturisasi model ER asli untuk mewakili hubungan yang benar antara entitas.

- *Chasm Traps*

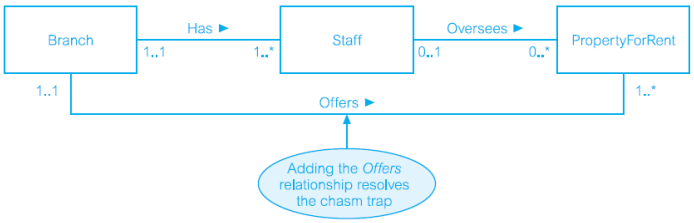
  Dimana model menunjukkan adanya hubungan antara jenis entitas, tetapi jalur tidak ada antara kejadian entitas tertentu.



Gambar 2.20 Contoh *Chasm Traps* (Connolly, T.M., et al. (2010))

Model ini merupakan fakta-fakta dimana satu cabang memiliki satu atau lebih staff yang mengawasi nol atau lebih properti untuk disewakan. Tidak semua staff

mengawasi properti, dan tidak semua properti diawasi oleh anggota staff. Masalah muncul ketika ingin tahu mana properti yang tersedia di masing-masing cabang.



Gambar 2.21 Contoh Restrukturisasi Untuk Menghapus *Chasm Traps* (Connolly, T.M., et al. (2010))

Mewakili relasi yang sebenarnya antara entitas tersebut. Model ini memastikan bahwa, setiap saat properti yang terkait dengan masing-masing cabang diketahui, termasuk properti yang belum dialokasikan untuk anggota staff.

2.1.9 *Enhanced Entity Relationship Modeling*

Menurut Connolly, T.M., et al. (2010), *enhanced entity relationship* adalah model ER yang mendukung konsep semantik tambahan. Berikut tiga konsep tambahan yang paling penting dan berguna dari model EER yaitu:

2.1.9.1 *Specialization/Generalization*

Specialization adalah proses memaksimalkan perbedaan antara anggota dari suatu entitas dengan mengidentifikasi karakteristik yang membedakan mereka. Spesialisasi adalah pendekatan top-down untuk mendefinisikan satu set *superclass* dan *subclass* yang terkait. Himpunan *subclass* didefinisikan berdasarkan beberapa karakteristik yang membedakan dari entitas dalam supernya.

Generalization adalah proses meminimalkan perbedaan antara entitas dengan mengidentifikasi karakteristik umum mereka. Proses generalisasi adalah

pendekatan bottom-up, yang menghasilkan identifikasi *superclass* umum dari jenis entitas asli.

Konsep spesialisasi / generalisasi dikaitkan dengan jenis khusus dari entitas yang dikenal sebagai *superclass* dan *subclass*, dan proses *attribute inheritance*.

Superclass adalah sebuah tipe entitas yang mencakup satu atau lebih sub kelompok yang berbeda dari kejadian, yang perlu diwakili dalam model data.

Subclass adalah sebuah pengelompokan yang berbeda kemunculannya dari suatu entitas, yang perlu diwakili dalam model data.

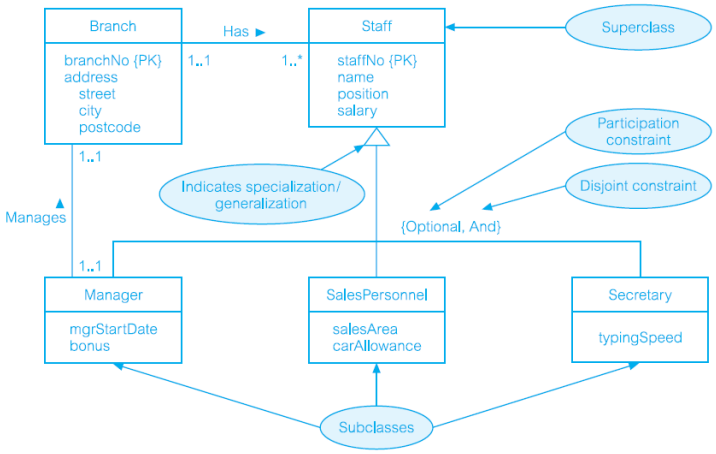
Ada dua *constraints* yang mungkin berlaku untuk spesialisasi / generalisasi yaitu:

- *Participation Constraint*

  Menentukan apakah setiap anggota *superclass* harus berpartisipasi sebagai anggota *subclass*.

- *Disjoint Constraint*

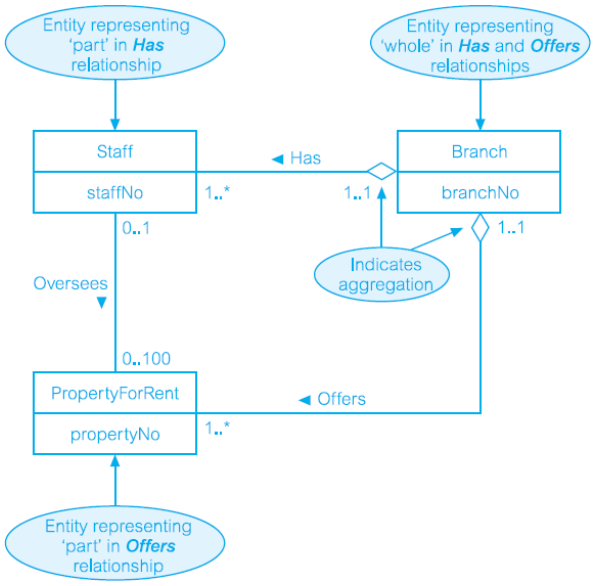
  Menggambarkan hubungan antara anggota *subclass* dan mengindikasikan apakah mungkin bagi seorang anggota *superclass* menjadi anggota dari satu atau lebih dari satu *subclass*.



Gambar 2.22 Contoh *Specialization*/*Generalization*, *Constraints* dan *Superclass*/*Subclass* (Connolly, T.M., et al. (2010))

2.1.9.2*Aggregation*

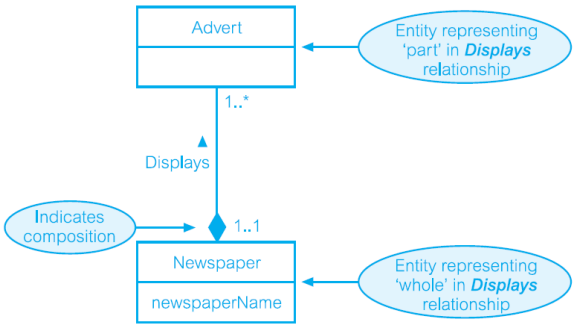
Aggregation adalah sebuah hubungan ‘memiliki sebuah’ atau ‘bagian dari’ antara tipe entitas di mana satu mewakili ‘keseluruhan’ dan yang lainnya ‘bagian’. Berikut contoh menampilkan dua agregasi, yaitu Branch memiliki Staff dan Branch menawarkan PropertyForRent. Dalam kedua hubungan, entitas Branch merepresentasikan ‘keseluruhan’ dan oleh karena itu bentuk *diamond* putih ditempatkan di samping entitas ini.



Gambar 2.23 Contoh *Aggregation* (Connolly, T.M., et al. (2010))

2.1.9.3*Composition*

Composition adalah sebuah bentuk spesifik dari agregasi yang merupakan hubungan antara entitas, di mana ada kepemilikan yang kuat dan secara kebetulan selamanya antara ‘keseluruhan’ dan ‘bagian’. Berikut contoh Newspaper menampilkan komposisi Advert, bentuk *diamond* gelap ditempatkan di sebelah entitas Newspaper yang merupakan ‘keseluruhan’ dalam hubungan ini.



Gambar 2.24 Contoh *Composition* (Connolly, T.M., et al. (2010))

2.1.10 *Normalization*

Menurut Connolly, T.M., et al. (2010), *normalization* adalah sebuah teknik untuk menghasilkan satu set relasi dengan properti yang diinginkan, mengingat kebutuhan data suatu perusahaan.

Menurut Rainer, R.K. & Cegielski, C.G. (2011), informasi adalah metode untuk menganalisis dan mengurangi database relasional untuk bentuk yang paling efisien untuk redundansi minimum, integritas data maksimum, dan kinerja pengolahan terbaik.

Tujuan dari normalisasi adalah untuk mengidentifikasi satu set yang cocok dari relasi yang mendukung kebutuhan data suatu perusahaan. Karakteristik dari set yang sesuai dari relasi meliputi berikut ini:

- Jumlah minimal atribut yang diperlukan untuk mendukung kebutuhan data perusahaan
- Atribut dengan relasi logis dekat (digambarkan sebagai ketergantungan fungsional) ditemukan dalam hubungan yang sama
- Redundansi minimal dengan setiap atribut diwakili hanya sekali dengan pengecualian penting dari atribut yang membentuk semua atau bagian dari *foreign keys* merupakan faktor penting untuk menggabungkan relasi terkait

Berikut proses-proses dalam normalisasi:

2.1.10.1 *Unnormalized Form* (UNF)

Unnormalized Form (UNF) adalah sebuah tabel yang berisi satu atau lebih kelompok yang berulang. Proses normalisasi dengan terlebih dahulu mentransfer data dari sumber (misalnya, entri data formulir standar) ke dalam format tabel dengan baris dan kolom. Dalam format ini, tabel dalam *Unnormalized Form* disebut sebagai tabel *unnormalized*.

ClientRental

clientNo	cName	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	John Kay	PG4	6 Lawrence St, Glasgow	1-Jul-03	31-Aug-04	350	CO40	Tina Murphy
		PG16	5 Novar Dr, Glasgow	1-Sep-04	1-Sep-05	450	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-02	10-June-03	350	CO40	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-03	1-Dec-04	375	CO93	Tony Shaw
		PG16	5 Novar Dr, Glasgow	1-Nov-05	10-Aug-06	450	CO93	Tony Shaw

Gambar 2.25 Contoh *Unnormalized Form* (UNF) (Connolly, T.M., et al. (2010))

2.1.10.2 *First Normal Form* (1NF)

First Normal Form (1NF) adalah suatu relasi di mana persimpangan setiap baris dan kolom berisi satu dan hanya satu nilai. Untuk mengubah tabel *unnormalized* ke 1NF dengan mengidentifikasi dan menghapus kelompok yang berulang dalam tabel. Sebuah kelompok berulang adalah atribut, atau kelompok atribut, dalam tabel yang terjadi dengan beberapa nilai untuk kejadian tunggal dari atribut kunci nominasi untuk tabel itu.

ClientRental

clientNo	propertyNo	cName	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	John Kay	6 Lawrence St, Glasgow	1-Jul-03	31-Aug-04	350	CO40	Tina Murphy
CR76	PG16	John Kay	5 Novar Dr, Glasgow	1-Sep-04	1-Sep-05	450	CO93	Tony Shaw
CR56	PG4	Aline Stewart	6 Lawrence St, Glasgow	1-Sep-02	10-Jun-03	350	CO40	Tina Murphy
CR56	PG36	Aline Stewart	2 Manor Rd, Glasgow	10-Oct-03	1-Dec-04	375	CO93	Tony Shaw
CR56	PG16	Aline Stewart	5 Novar Dr, Glasgow	1-Nov-05	10-Aug-06	450	CO93	Tony Shaw

Gambar 2.26 Contoh *First Normal Form* (1NF) (Connolly, T.M., et al. (2010))

2.1.10.3 *Second Normal Form* (2NF)

Second Normal Form (2NF) adalah suatu relasi yang ada di 1NF dan setiap atribut *non primary key* sepenuhnya secara fungsional tergantung pada *primary key*. Normalisasi relasi 1NF ke 2NF melibatkan penghapusan ketergantungan parsial. Jika ketergantungan parsial ada, selanjutnya menghapus atribut ketergantungan parsial dari relasi dengan menempatkan mereka dalam hubungan baru bersama dengan salinan determinan mereka.

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-03	31-Aug-04
CR76	PG16	1-Sep-04	1-Sep-05
CR56	PG4	1-Sep-02	10-Jun-03
CR56	PG36	10-Oct-03	1-Dec-04
CR56	PG16	1-Nov-05	10-Aug-06

PropertyOwner

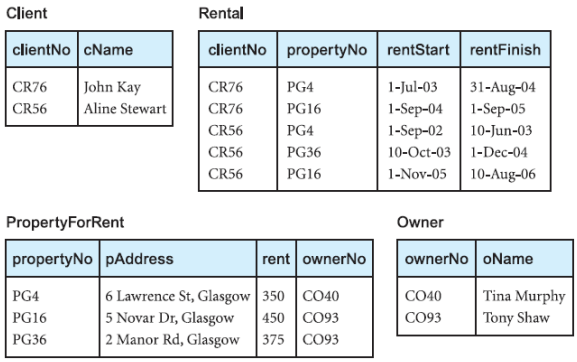
propertyNo	pAddress	rent	ownerNo	oName
PG4	6 Lawrence St, Glasgow	350	CO40	Tina Murphy
PG16	5 Novar Dr, Glasgow	450	CO93	Tony Shaw
PG36	2 Manor Rd, Glasgow	375	CO93	Tony Shaw

Gambar 2.27 Contoh *Second Normal Form* (2NF) (Connolly, T.M., et al. (2010))

2.1.10.4 *Third Normal Form* (3NF)

Third Normal Form (3NF) adalah suatu relasi yang ada di 1NF dan 2NF di mana tidak ada atribut *non primary key* adalah secara transitif bergantung pada *primary key*.

Normalisasi relasi 2NF ke 3NF melibatkan penghapusan ketergantungan transitif. Jika ketergantungan transitif ada, selanjutnya menghapus atribut ketergantungan secara transitif dari relasi dengan menempatkan atribut dalam hubungan baru bersama dengan salinan determinan.



Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-03	31-Aug-04
CR76	PG16	1-Sep-04	1-Sep-05
CR56	PG4	1-Sep-02	10-Jun-03
CR56	PG36	10-Oct-03	1-Dec-04
CR56	PG16	1-Nov-05	10-Aug-06

PropertyForRent

propertyNo	pAddress	rent	ownerNo
PG4	6 Lawrence St, Glasgow	350	CO40
PG16	5 Novar Dr, Glasgow	450	CO93
PG36	2 Manor Rd, Glasgow	375	CO93

Owner

ownerNo	oName
CO40	Tina Murphy
CO93	Tony Shaw

Gambar 2.28 Contoh *Third Normal Form* (3NF) (Connolly, T.M., et al. (2010))

2.1.11 *Database Design Methodology*

Menurut Connolly, T.M., et al. (2010), *design methodology* adalah pendekatan terstruktur yang menggunakan prosedur, teknik, alat, dan alat bantu dokumentasi untuk mendukung dan memfasilitasi proses desain.

Sebuah metodologi desain terdiri dari tahapan masing-masing berisi sejumlah langkah yang memandu desainer dalam teknik yang tepat pada setiap tahapan proyek. Metodologi desain juga membantu desainer untuk merencanakan, mengelola, mengendalikan, dan mengevaluasi proyek-proyek pengembangan *database*. Selain itu, pendekatan terstruktur untuk menganalisis dan pemodelan satu set persyaratan untuk *database* dengan cara yang standar dan terorganisir.

Dalam menyajikan metodologi desain database ini, proses desain dibagi menjadi tiga tahap utama yaitu:

2.1.11.1 *Conceptual Database Design*

Conceptual Database Design adalah proses membangun sebuah model dari data yang digunakan dalam

suatu perusahaan desain *database*, independen dari semua pertimbangan fisik.

Fase *conceptual database design* dimulai dengan penciptaan model data konseptual dari perusahaan, yang sepenuhnya independen dari rincian implementasi seperti DBMS sasaran, program aplikasi, bahasa pemrograman, platform perangkat keras, masalah performa, atau pertimbangan fisik lainnya.

Panduan langkah demi langkah untuk desain *database* konseptual:

- *Build Conceptual Data Model*

  Untuk membangun sebuah model data konseptual dari kebutuhan data perusahaan.

  - *Identify entity types*

    Untuk mengidentifikasi tipe entitas yang dibutuhkan.
  - *Identify relationship types*

    Untuk mengidentifikasi relasi penting yang ada antara tipe entitas.
  - *Identify and associate attributes with entity or relationship types*

    Untuk mengaitkan atribut dengan entitas atau jenis relasi yang sesuai.
  - *Determine attribute domains*

    Untuk menentukan domain untuk atribut dalam model data konseptual.
  - *Determine candidate, primary, and alternate key attributes*

    Untuk mengidentifikasi *candidate key* untuk setiap jenis entitas dan jika ada lebih dari satu *candidate key*, untuk memilih salah satu untuk menjadi *primary key* dan yang lain sebagai *alternate key*.
  - *Consider use of enhanced modeling concepts (optional step)*

Untuk mempertimbangkan penggunaan konsep pemodelan peningkatan seperti spesialisasi / generalisasi, agregasi, dan komposisi.

- *Check model for redundancy*

  Untuk memeriksa keberadaan setiap redundansi dalam model.

- *Validate conceptual model against user transactions*

  Untuk memastikan bahwa model konseptual mendukung transaksi yang diperlukan.

- *Review conceptual data model with user*.

  Untuk meninjau model data konseptual dengan pengguna untuk memastikan bahwa mereka mempertimbangkan model untuk menjadi representasi yang benar dari persyaratan data perusahaan.

2.1.11.2 *Logical Database Design*

Logical Database Design adalah proses membangun sebuah model dari data yang digunakan dalam suatu perusahaan berdasarkan pada model data yang spesifik, tetapi independen dari DBMS tertentu dan pertimbangan fisik lainnya.

Fase *logical database design* memetakan model konseptual ke model logis yang dipengaruhi oleh model data untuk *database* target (misalnya, model relasional). Model data logis merupakan sumber informasi untuk fase desain fisik, menyediakan perancang *database* fisik dengan kendaraan untuk membuat manfaat yang sangat penting bagi desain *database* yang efisien.

Panduan langkah demi langkah untuk desain *database* logikal:

- *Build Logical Data Model*

  Untuk menerjemahkan model data konseptual menjadi model data logis dan kemudian untuk memvalidasi model ini untuk memeriksa bahwa secara struktural benar dan mampu mendukung transaksi yang dibutuhkan.

  - *Derive relations for logical data model*

    Untuk menciptakan relasi untuk model data logis untuk mewakili entitas, relasi, dan atribut yang telah diidentifikasi.
  - *Validate relations using normalization*

    Untuk memvalidasi relasi dalam model data logis menggunakan normalisasi.
  - *Validate relations against user transactions*

    Untuk memastikan bahwa relasi dalam model data logis mendukung transaksi yang dibutuhkan.
  - *Check integrity constraints*

    Untuk memeriksa batasan integritas terwakili dalam model data logis.
  - *Review logical data model with user*

    Untuk meninjau model data logis dengan pengguna untuk memastikan bahwa mereka mempertimbangkan model untuk menjadi representasi benar dari persyaratan data perusahaan.
  - *Merge logical data models into global model (optional step)*

    Untuk menggabungkan model data logis lokal ke dalam model data logis global tunggal yang mewakili semua pandangan pengguna *database.*
  - *Check for future growth*

    Untuk menentukan apakah ada kemungkinan perubahan yang signifikan di masa mendatang dan

untuk menilai apakah model data logis dapat mengakomodasi perubahan ini.

2.1.11.3 *Physical Database Design*

Physical Database Design adalah proses memproduksi penjelasan implementasi *database* pada *secondary storage*, menggambarkan hubungan dasar, organisasi file, indeks yang digunakan untuk mencapai akses yang efisien terhadap data, dan setiap kendala integritas terkait dan langkah-langkah keamanan.

Fase *physical database design* memungkinkan desainer untuk membuat keputusan tentang bagaimana *database* diimplementasikan. Oleh karena itu, desain fisik disesuaikan dengan DBMS tertentu. Ada umpan balik antara desain fisik dan logis, karena keputusan yang diambil selama desain fisik untuk meningkatkan kinerja dapat mempengaruhi model data logis.

Panduan langkah demi langkah untuk desain *database* fisikal:

- *Translate logical data model for target DBMS*

  Untuk menghasilkan skema database relasional dari model data logis yang dapat diimplementasikan dalam DBMS sasaran.

  - *Design base relations*

    Untuk memutuskan bagaimana untuk menampilkan relasi dasar yang diidentifikasi dalam model data logis dalam DBMS sasaran.

  - *Design representation of derived data*

    Untuk memutuskan bagaimana untuk menampilkan data yang diperoleh ada dalam model data logis dalam DBMS sasaran.

  - *Design general constraints*

    Untuk merancang kendala umum untuk DBMS sasaran.

- *Design file organizations and indexes*

  Untuk menentukan file organisasi yang optimal untuk menyimpan relasi dasar dan indeks yang diperlukan untuk mencapai kinerja yang dapat diterima, yaitu cara di mana relasi dan tupel akan diadakan pada penyimpanan sekunder.

  - *Analyze transactions*

    Untuk memahami fungsi dari transaksi yang akan berjalan pada *database* dan untuk menganalisis transaksi penting.

  - *Choose file organizations*

    Untuk menentukan sebuah file organisasi yang efisien untuk setiap relasi dasar.

  - *Choose indexes*

    Untuk menentukan apakah menambahkan indeks akan meningkatkan kinerja sistem.

  - *Estimate disk space requirements*

    Untuk memperkirakan jumlah ruang disk yang akan dibutuhkan oleh *database.*

- *Design user views*

  Untuk merancang user views yang telah diidentifikasi selama tahap pengumpulan kebutuhan dan analisis siklus pengembangan sistem *database*.

- *Design security mechanisms*

  Untuk merancang mekanisme keamanan bagi *database* seperti yang ditentukan oleh pengguna selama tahap persyaratan dan pengumpulan siklus pengembangan sistem *database*.

- *Consider the introduction of controlled redudancy*

  Untuk menentukan apakah mengenalkan redundansi dengan cara yang terkendali dengan relaksasi aturan normalisasi akan meningkatkan kinerja sistem.

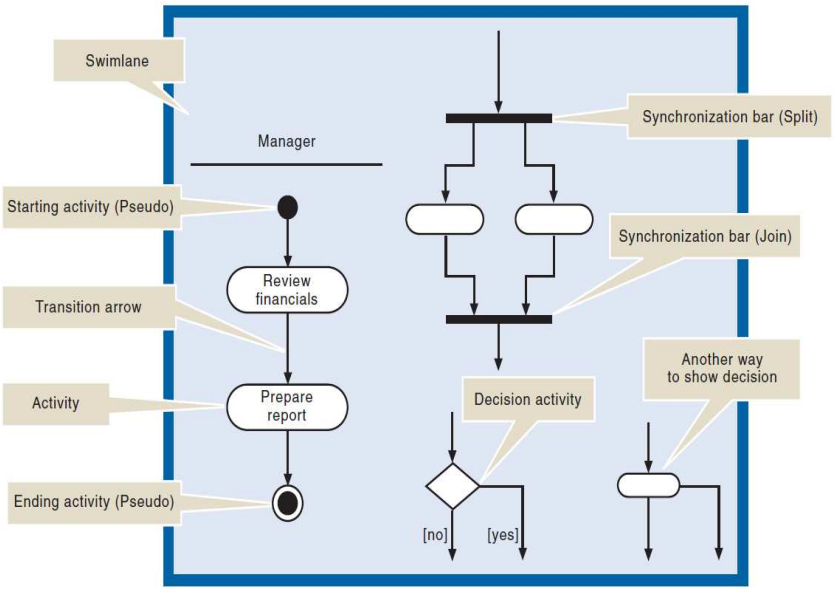
- *Monitor and tune the operational system*

Untuk memonitor sistem operasional dan meningkatkan kinerja sistem untuk memperbaiki keputusan desain yang tidak pantas atau mencerminkan perubahan kebutuhan.

2.1.12 *Activity Diagram*

Menurut Satzinger, J.W., et al. (2010), *activity diagram* adalah sebuah diagram alur kerja yang menggambarkan berbagai aktivitas pengguna atau sistem, orang yang melakukan setiap kegiatan, dan aliran sekuensial. *Activity diagram* adalah salah satu diagram *Unified Modelling Language* yang terkait dengan pendekatan berorientasi objek, tetapi dapat digunakan untuk pendekatan pembangunan.

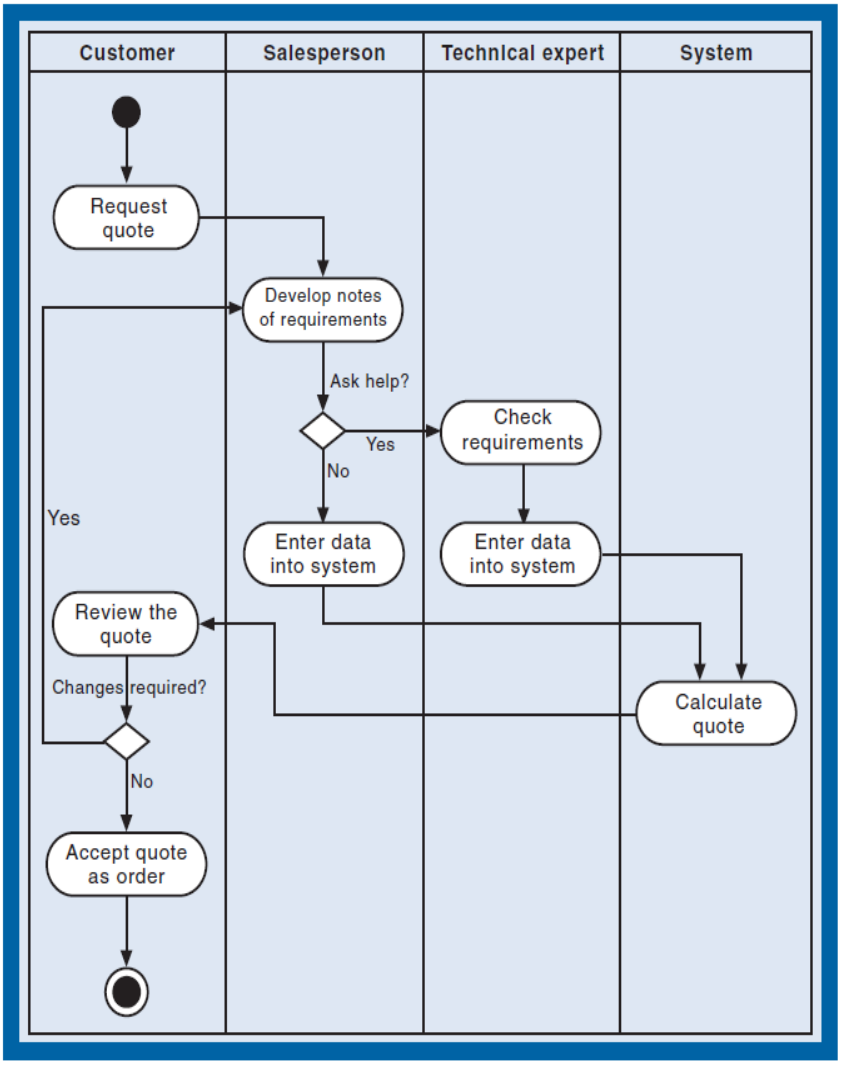
Berikut gambar simbol-simbol yang digunakan dalam *activity diagram* beserta artinya:



Gambar 2.29 Simbol-simbol *Activity Diagram* (Satzinger, J.W., et al. (2010))

- *Swimlane*: adalah area persegi pada *activity diagram* untuk mewakili seseorang yang melakukan kegiatan.
- *Initial state*: untuk memulai *activity diagram.*
- *Transition arrow*: untuk menampilkan aliran aktivitas.
- *Activity*: untuk menampilkan aktivitas.
- *Syncronization bar (split)*: adalah simbol yang digunakan dalam sebuah *activity diagram* untuk mengontrol pembagian atau penyatuan jalur berurutan, split untuk membagi satu menjadi dua aliran-aliran dalam kegiatan waktu yang sama.
- *Syncronization bar (join)*: adalah simbol yang digunakan dalam *activity diagram* untuk mengontrol pembagian atau penyatuan jalur berurutan, join adalah untuk menggabungkan dua aliran dari aktivitas menjadi satu aliran kegiatan.
- *Decision activity*: adalah simbol untuk membuat keputusan dalam aliran berupa ya atau tidak.
- *Final state*: untuk mengkhiri aktivitas.

Berikut contoh pengambaran *activity diagram*:



Gambar 2.30 Contoh *Activity Diagram* (Satzinger, J.W., et al. (2010))

Gambar 2.29 adalah diagram aktivitas sebenarnya untuk alur kerja. Alur kerja ini merupakan seorang pelanggan meminta kutipan dari seorang tenaga penjualan. Jika permintaan yang sederhana, penjual dapat memasukkan data dan membuat kutipan. Jika kompleks, penjual meminta bantuan dari ahli teknis untuk menghasilkan kutipan. Dalam kedua kasus, sistem komputer menghitung rincian dari kutipan.

2.2 Teori Khusus

2.2.1 Sistem *Housekeeping*

Pengertian *housekeeping* menurut Andrews, S. (2013), *housekeeping* adalah pusat dukungan fundamental dalam pembentukan perhotelan. Di hotel, *housekeeping* umumnya mempekerjakan sebagian besar orang dan memiliki biaya tertinggi dalam departemen apapun. Layanan pelanggan, lokasi, dan fasilitas bisa mendatangkan tamu tapi tanpa kebersihan, tidak akan pernah ada mendatangi tamu lagi atau ucapan positif. Fungsi dari Departemen *housekeeping* oleh karenanya penting tetapi sering diterima begitu saja.

Pengertian *housekeeping* menurut Walker, J.W. (2010), *housekeeping* adalah departemen terbesar dalam hal jumlah orang yang dipekerjakan. Sampai dengan 50 persen dari karyawan hotel dapat bekerja di departemen ini. Karena kerja keras dan bayaran yang relatif rendah, perputaran karyawan sangat tinggi di departemen penting ini.

2.2.2 Sistem *Sales*/Penjualan

Pengertian penjualan menurut Kotler, P., & Keller, K.L. (2012), penjualan adalah mendekati, menyajikan, menjawab pertanyaan, mengatasi keberatan, dan menutup penjualan.

Menurut Pinar, M., Hardin, J. R., & Eser, Z. (2011), penjualan adalah suatu proses dengan tahap berturut-turut, yang meliputi pertemuan prospek, membuat presentasi, menjawab pertanyaan, mengatasi penolakan, upaya uji coba yang dekat, dan menutup penjualan.

Menurut Age, L. J. (2011), penjualan adalah proses fenomena pada dasarnya kompleks karena pada akhirnya tergantung pada pemecahan masalah dalam konteks interaksi manusia pribadi.

2.2.3 Sistem *Reception*

Pengertian *reception* menurut Sudhir Andrews, S. (2013), *reception* adalah mencatatakan pemesanan, menangani janji dan menangani penagihan, dimana sebagai eksekutif bisnis yang tinggal di lantai eksekutif hotel memiliki akses gratis ke lounge bisnis yang memiliki suasana eksklusif.

Pengertian *reception* menurut Andrews, S. (2010), *reception* adalah bagian depan *front office*. Ini adalah tempat di mana tamu membangun kesan bagi para tamu untuk tetap tinggal.

Pengertian *reception* menurut Heiser, S., Stickler, U., & Furnborough, C. (2013), *reception* adalah bagian yang bertanggung jawab untuk menyiapkan dan memberi label penembusan kamar untuk kelompok-kelompok kamar baru.

2.2.4 Sistem Administrasi

Pengertian administrasi menurut Sapru, R.K. (2013), administrasi adalah kegiatan yang berkaitan dengan menjaga catatan dan pengolahan informasi, dokumen dan kegiatan yang bersangkutan dengan menerapkan aturan, prosedur dan kebijakan yang ditentukan oleh orang lain.

Pengertian administrasi menurut Denhardt, R.B., & Catlaw, T.J. (2015) administrasi adalah perhatian dengan bagaimana organisasi harus dibangun dan dioperasikan untuk menyelesaikan pekerjaan secara efisien.

2.2.5 *Hospitality*

Pengertian *hospitality* menurut Barrows, C.W., Powers, T., & Reynolds, D. (2012), *hospitality* adalah penerimaan dan menghibur tamu, pengunjung atau orang asing dengan kemurahan dan niat baik. Perhotelan berasal dari kata *hospice*(rumah sakit), istilah untuk sebuah rumah abad pertengahan istirahat bagi wisatawan dan peziarah.

Pengertian *hospitality* menurut Walker, J.W. (2010), *hospitality* adalah sesuatu yang menarik, menyenangkan dan

menstimulasi di mana untuk menikmati karir, ditambah lagi mendapatkan kompensasi cukup baik dan memiliki peluang kemajuan yang sangat baik.

Pengertian *hospitality* menurut Chon, Kaye(Kye-Sung)., & Maier, T. (2010), *hospitality* adalah menerima tamu dengan cara yang murah hati dan ramah, menciptakan lingkungan yang menyenangkan atau mempertahankan, memuaskan kebutuhan tamu, mengantisipasi keinginan tamu, menghasilkan suasana yang ramah dan aman.

Pengertian *hospitality* menurut Bharwani, S., & Jauhari, V. (2013), *hospitality* adalah hubungan yang didasarkan pada tuan rumah dan tamu yang menekankan bahwa hubungan tuan rumah dan tamu yang merupakan ciri khas utama perhotelan dari beberapa dimensi lain yang muncul.

2.2.6 *Hospitality Management*

Pengertian *hospitality management* menurut Barrows, C.W., Powers, T., & Reynolds, D. (2012), *hospitality management* adalah pengelolaan dalam penerimaan dan menghibur tamu, pengunjung atau orang asing dengan kemurahan dan niat baik. Perhotelan berasal dari kata *hospice*(rumah sakit), istilah untuk sebuah rumah abad pertengahan istirahat bagi wisatawan dan peziarah.

2.2.7 Bahasa Pemrograman C#

Pengertian bahasa pemrograman c# menurut msdn.microsoft.com (2013), c# adalah sebuah bahasa pemrograman yang dirancang untuk membangun berbagai aplikasi yang berjalan di .NET Framework. C# sederhana, kuat, jenis yang aman, dan berorientasi objek. Banyak inovasi di C# memungkinkan pengembangan aplikasi yang cepat sementara tetap mempertahankan ekspresi dan keanggunan gaya bahasa C.

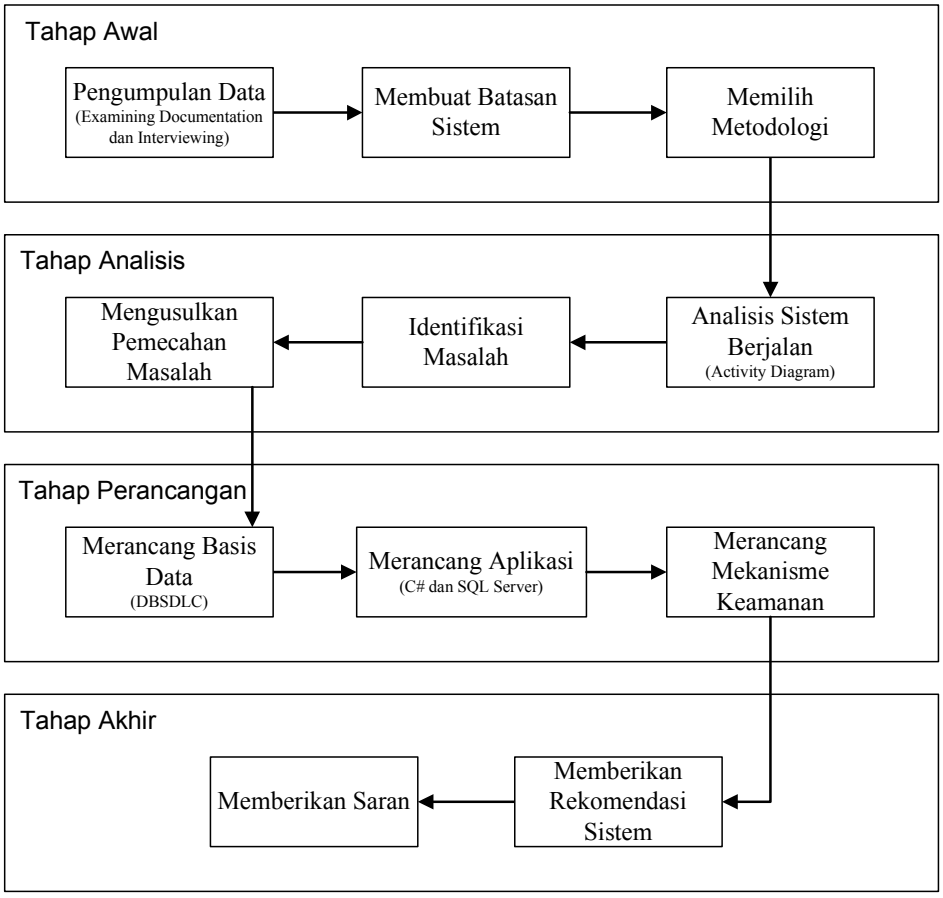
Visual C# merupakan implementasi dari bahasa C# oleh Microsoft. Visual Studio mendukung Visual C# dengan editor dengan fitur lengkap kode, compiler, proyek template, desainer, *code wizards*, debugger kuat dan mudah digunakan, dan alat-alat lainnya. *Class*

library .NET Framework menyediakan akses ke banyak layanan sistem operasi dan berguna, kelas-kelas lain yang dirancang dengan baik yang mempercepat siklus pengembangan signifikan.

2.2.8 *Structured Query Language*

Menurut Connolly, T.M., et al. (2010), *structured query language* adalah bahasa non-prosedural, yang terdiri dari kata-kata bahasa Inggris standar seperti SELECT, INSERT, DELETE, yang dapat digunakan oleh para profesional dan non-profesional. Keduanya baik formal dan de facto bahasa standar untuk mendefinisikan dan memanipulasi database relasional.

2.3 Kerangka Berpikir



Gambar 2.31 Kerangka Berpikir