

SKRIPSI

KONVERSI JADWAL MENGAWAS UJIAN KE FORMAT ICS
DENGAN APACHE POI, ICAL4J, DAN JAVAFX



ARIQ RAHMAERI

NPM: 2011730066

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2015

UNDERGRADUATE THESIS

**CONVERSION OVERSEEN EXAM SCHEDULE TO ICS
FORMAT WITH APACHE POI, ICAL4J, AND JAVAFX**



ARIQ RAHMAERI

NPM: 2011730066

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2015**

ABSTRAK

Setiap tahun dosen FMIPA UNPAR menerima *printout* jadwal mengawas ujian yang dibuat dengan excel. Walaupun datanya bersifat digital, namun lebih baik jika data tersebut dibuat terstruktur sehingga dapat dibaca oleh mesin. Dari permasalahan tersebut maka dalam tugas akhir ini akan dibahas tentang pengembangan suatu program yang dapat membaca data excel tersebut dan merubahnya dalam format calendar digital atau biasa disebut .ics, sehingga dosen dapat memasukan jadwal mengawas ujian kedalam gawai pribadinya. Program ini akan menggunakan tiga bahasa pemrograman yaitu Apache POI, iCal4j, dan Java FX. Apache ROI bertugas membaca struktur data excel sehingga dapat dibaca oleh program, Java FX berfungsi sebagai *interface* program, dan Ical4j bertugas mengkonversi data yang telah dibaca program kedalam format iCalendar atau .ics .

Kata-kata kunci: Apache POI, iCal4j, Java FX

ABSTRACT

Every year lecture of FMIPA UNPAR get printout schedule of invigilation in excel format. Although, the excel data is digital data, but it's better if the data is structured so can be read by machine. Based of the problem above, then this thesis will be discussing about developing program that can read invigilation schedule in excel format and converting the schedule to digital calendar format or commonly called .ics, so that lecture can import the schedule to their personal gadget. This software will be build based on three programming language, that is Apache POI, iCal4j, and Java FX. Apache POI will be handle reading input data so the data can be imported to software, Java FX is functionate as interface of the software, and iCal4j handles of converting previously read data to iCalendar format or .ics .

Keywords: Apache POI, iCal4j, Java FX

DAFTAR ISI

DAFTAR ISI	ix
DAFTAR GAMBAR	x
DAFTAR TABEL	xi
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	1
1.3 Tujuan	1
1.4 Batasan Masalah	2
1.5 Metodologi Penelitian	2
1.6 Sistematika Pembahasan	2
2 DASAR TEORI	5
2.1 Decision Support Systems	5
2.2 Business Intelligence	5
2.2.1 Arsitektur business intelligence	5
2.3 Data Warehouse	6
2.3.1 Definisi <i>Data Warehouse</i>	7
2.3.2 Arsitektur <i>Data Warehouse</i>	8
2.3.3 Proses ETCL	10
2.3.4 Struktur <i>Data Warehouse</i> Berdasarkan Konsep Data Multidimensi	13
2.3.5 Teknik Pemodelan Dimensional	15
2.3.6 Hubungan OLAP, <i>Data warehouse</i> , DSS, dengan <i>Business Intelligence</i>	16
2.4 <i>Model View Controller</i> (MVC)	17
2.5 <i>Object Relational Mapping</i> (ORM)	18
2.6 Framework OODO	19
2.6.1 Arsitektur OODO	19
3 ANALISIS	25
3.1 Analisis Kebutuhan dan Fitur Perangkat Lunak	25
3.1.1 Analisis Kebutuhan	25
3.1.2 Fitur Perangkat Lunak	25
3.1.3 Diagram Aktifitas pada BI Tool	26
3.2 Permodelan Tool	34
3.3 Pemodelan Basisdata	37
3.3.1 Perancangan Basisdata	37
4 PERANCANGAN	39
4.1 Perancangan Lojik Basisdata	39
4.2 Perancangan Fisik Basisdata	40
4.3 Diagram Kelas	45

5 IMPLEMENTASI DAN PENGUJIAN	53
5.1 Lingkungan Implementasi	53
5.1.1 Lingkungan Perangkat Keras	53
5.1.2 Lingkungan Perangkat Lunak	53
5.2 Implementasi Tabel Basisdata	53
5.2.1 Tabel Basisdata ETL dan Bisnis	53
5.3 Implementasi Kode Program Phyton	61
5.4 Implementasi Antar Muka	62
5.4.1 Implementasi Antar Muka untuk <i>Business</i>	62
5.4.2 Implementasi Antar Muka untuk Proses ETL	62
5.5 Pengujian	68
6 KESIMPULAN DAN SARAN	71
6.1 Kesimpulan	71
6.2 Saran	71
DAFTAR REFERENSI	73
A THE PROGRAM	75
B THE SOURCE CODE	77
C THE SOURCE CODE	79

DAFTAR GAMBAR

2.1	Komponen utama dari sistem <i>business intelligence</i> [1]	6
2.2	Perbedaan OLTP dan OLAP sistem [1]	7
2.3	Gambar <i>Single-Layer Architecture</i> [2]	8
2.4	Gambar <i>Two-Layer Architecture</i> [2]	9
2.5	Gambar <i>Three-Layer Architecture</i> [2]	10
2.6	Contoh Proses ETCL Pada <i>Data Warehouses</i> [2]	11
2.7	Contoh Proses Transformasi Pada <i>Data Warehouses</i> [2]	12
2.8	Contoh <i>Three Dimensional Cube</i> [2]	13
2.9	<i>Drill-Down</i> [2]	14
2.10	<i>Roll-Up</i> [2]	14
2.11	<i>Slice</i> (Atas) dan <i>Dice</i> (Bawah)[2]	15
2.12	<i>Star Schema</i> [2]	15
2.13	<i>Snowflakes Schema</i> [2]	16
2.14	<i>Constellation Schema</i> [2]	17
2.15	Diagram MVC	17
2.16	Konsep ORM	18
2.17	Arsitektur OODO	19
2.18	Arsitektur OODO	21
2.19	Diagram MVC	23
3.1	Fitur BI <i>tool</i>	26
3.2	Prosedur Membuat Bisnis Baru	26
3.3	Prosedur Memasukan <i>Data Source</i>	27
3.4	Prosedur Menentukan Tipe dan Format Data	28
3.5	Prosedur Melakukan <i>Lookup</i>	29
3.6	Prosedur Melakukan <i>Sorting Data</i>	30
3.7	Prosedur Menambah atau Memodifikasi Kolom	30
3.8	Prosedur Mengaplikasikan Phyton <i>Script</i>	31
3.9	Prosedur Melakukan <i>Aggregate</i>	32
3.10	Prosedur Melakukan <i>Merge Join</i> dari dua <i>data source</i>	32
3.11	Prosedur Menyimpan Data dalam <i>Database</i>	33
3.12	Diagram use case BI <i>tool</i>	34
3.13	Struktur tabel BI <i>tool</i>	37
4.1	Diagram Relational pada Basisdata BI <i>tool</i>	40
4.2	Struktur kelas diagram BI <i>tool</i>	51
5.1	Implementasi tabel ba_business	54
5.2	Implementasi tabel ba_etl_header	54
5.3	Implementasi tabel ba_etl_detail	55
5.4	Implementasi tabel ba_etl_detail2	55
5.5	Implementasi tabel ba_etl_detail3	55
5.6	Implementasi tabel ba_etl_columns	56

5.7	Implementasi tabel ba_etl_columns_mapping	56
5.8	Implementasi tabel ba_etl_columns_mapping	57
5.9	Implementasi tabel ba_etl_sort_data_columns	57
5.10	Implementasi tabel ba_etl_derived_column_lines	58
5.11	Implementasi tabel ba_etl_lookup_fixed_lines	58
5.12	Implementasi tabel ba_etl_save_data_columns	59
5.13	Implementasi tabel ba_etl_lookup_model_mappings	59
5.14	Implementasi tabel ba_etl_merge_columns1	60
5.15	Implementasi tabel ba_etl_merge_columns2	60
5.16	Implementasi tabel ba_etl_aggregate_columns	61
5.17	kode program <i>load columns from source</i>	61
5.18	Kode Program untuk <i>mapping column</i>	62
5.19	Tampilan untuk membuat bisnis baru	62
5.20	Tampilan untuk membuat ETL baru	63
5.21	Tampilan untuk <i>upload source</i>	63
5.22	Tampilan untuk ubah data dan format data	64
5.23	Tampilan untuk menerapkan skrip phyton	64
5.24	Tampilan untuk <i>lookup</i> dari <i>existing table</i>	65
5.25	Tampilan untuk <i>lookup</i> dari <i>value</i> yang dituliskan pengguna	65
5.26	Tampilan untuk menambah atau memodifikasi kolom	66
5.27	Tampilan untuk <i>merging</i> dua <i>source</i>	66
5.28	Tampilan untuk melakukan <i>sort</i>	67
5.29	Tampilan untuk melakukan agregat	67
5.30	Tampilan untuk menyimpan kedalam <i>database</i>	68
A.1	Interface of the program	75

DAFTAR TABEL

4.1	Tabel <i>ba_bussiness</i>	40
4.2	Tabel <i>ba_etl_header</i>	41
4.3	Tabel <i>ba_etl_detail</i>	42
4.4	Tabel <i>ba_etl_columns</i>	42
4.5	Tabel <i>ba_etl_columns_mapping</i>	43
4.6	Tabel <i>ba_etl_format_data_lines</i>	43
4.7	Tabel <i>ba_etl_sort_data_columns</i>	43
4.8	Tabel <i>ba_etl_derived_column_lines</i>	44
4.9	Tabel <i>ba_etl_lookup_fixed_lines</i>	44
4.10	Tabel <i>ba_etl_save_data_column</i>	44
4.11	Tabel <i>ba_etl_lookup_model_mappings</i>	44
4.12	Tabel <i>ba_etl_merge_columns_1</i>	45
4.13	Tabel <i>ba_etl_merge_columns_2</i>	45
4.14	Tabel <i>ba_etl_aggregate_columns</i>	45
4.15	Tabel Kelas <i>ba_bussiness</i>	45
4.16	Tabel Kelas <i>ba_etl_header</i>	46
4.17	Tabel Kelas <i>ba_etl_columns</i>	46
4.18	Tabel Kelas <i>ba_etl_detail</i>	47
4.19	Tabel Kelas <i>ba_etl_columns_mapping</i>	48
4.20	Tabel Kelas <i>ba_etl_format_data_lines</i>	48
4.21	Tabel Kelas <i>ba_etl_sort_data_columns</i>	48
4.22	Tabel Kelas <i>ba_etl_derived_column_lines</i>	48
4.23	Tabel Kelas <i>ba_etl_save_data_columns</i>	48
4.24	Tabel Kelas <i>ba_etl_lookup_fixed_lines</i>	49
4.25	Tabel Kelas <i>ba_etl_lookup_model_mappings</i>	49
4.26	Tabel Kelas <i>ba_etl_merge_columns1</i>	49
4.27	Tabel Kelas <i>ba_etl_merge_columns2</i>	49
4.28	Tabel Kelas <i>ba_etl_aggregate_columns</i>	49
4.29	Tabel Kelas <i>ba_run_etl</i>	50
5.1	Contoh Tabel <i>customer</i>	68
5.2	Tabel <i>ba_bussiness</i>	69

¹ **BAB 1**

² **PENDAHULUAN**

³ **1.1 Latar Belakang**

⁴ Jadwal mengawas ujian merupakan hal yang rutin dibagikan kepada dosen setiap tengah
⁵ atau pada akhir semester. Setiap dosen menerima jadwal tersebut yang diterima dari ta-
⁶ ta usaha jurusan. Jadwal tersebut di *printout* oleh tata usaha jurusan sebelum dibagikan
⁷ kepada dosen. Format jadwal mengawas ujian di FTIS bersifat umum, dalam arti jadwal
⁸ tersebut menyimpan nama semua dosen yang mengawas, nama matakuliah, dan tempat pe-
⁹ laksanaan ujian. Dosen diharuskan melihat satu persatu baris untuk mendapatkan informasi
¹⁰ mengenai waktu, nama mata kuliah, dan tanggal dosen tersebut mengawas. Walaupun
¹¹ jadwal mengawas ujian telah berupa excel, namun tetap dirasa kurang efisien karena tidak
¹² tersusun berdasarkan dosen yang mengawas dan memungkinkan terjadinya kesalahan dalam
¹³ membaca jadwal oleh dosen. Alangkah lebih baik jika data tersebut dibuat lebih terstruk-
¹⁴ tur sehingga dapat dibaca oleh *software* dan dapat di integrasikan ke masing-masing gawai
¹⁵ dari dosen sehingga memudahkan dosen untuk melihat jadwal mengawas ujian dimanapun.

¹⁶ Dari penjelasan diatas, tugas akhir ini dimaksudkan untuk memudahkan dosen untuk
¹⁷ melihat jadwal mengawas ujian dimanapun dan kapanpun. Pengembangan perangkat lunak
¹⁸ ini menggunakan tiga bahasa pemograman, yaitu Apache POI, Java FX, dan iCal4j.

¹⁹ **1.2 Rumusan Masalah**

²⁰ Berdasarkan penjelasan latar belakang maka dapat dipaparkan rumusan masalah sebagai
²¹ berikut :

- ²² 1. Bagaimana perangkat lunak dapat membaca excel dengan Apache POI ?
- ²³ 2. Bagaimana membuat tampilan *user-friendly* dengan Java FX ?
- ²⁴ 3. Bagaimana perangkat lunak mengkonversi jadwal mengawas menjadi iCalendar dengan
²⁵ iCal4j ?

²⁶ **1.3 Tujuan**

²⁷ Tujuan dari karya ilmiah ini dapat dipaparkan sebagai berikut:

- ²⁸ 1. Mempelajari kemampuan Apache POI, Java FX, dan iCal4j.
- ²⁹ 2. Membuat perangkat lunak yang mampu membaca file excel yang dipublikasikan oleh
³⁰ tata usaha, dan mengkonversi dalam bentuk iCalendar dengan filter tertentu.
- ³¹ 3. Memudahkan dosen dalam mendapatkan jadwal yang dapat di integrasikan dengan
³² aplikasi iCalendar sehingga dapat melihat jadwal dimanapun dan kapanpun.

1 1.4 Batasan Masalah

2 Batasan masalah dalam penelitian ini agar dapat fokus pada pengembangan perangkat lunak
3 konversi jadwal mengawas ujian :

- 4 ● Data *input* PL merupakan file excel dari TU yang telah dimodifikasi dan distandardisasi
5 penulisannya agar dapat dibaca oleh PL.
- 6 ● PL memiliki minimal satu *filter*.
- 7 ● PL bersifat *offline*
- 8 ● Format *output* PL adalah iCalendar atau .ics .

9 1.5 Metodologi Penelitian

10 Untuk menunjang penelitian maka diperlukan data untuk pengujian maupun pengetahuan
11 teori yang akan diterapkan. Berikut adalah kegiatan yang akan dilakukan:

- 12 1. Melakukan studi pustaka mengenai:
- 13 ● Apache POI
- 14 ● Java FX
- 15 ● iCal4j
- 16 ● Konsep MVC
- 17 ● memperdalam Netbeans
- 18 2. Melakukan standarisasi data *input*.
- 19 3. Melakukan analisa kebutuhan dan fitur yang dimiliki perangkat lunak.
- 20 4. Melakukan perancangan yang terdiri dari ER-diagram, use case dan *user interface*.
21 Serta mengimplementasikan rancangan tersebut kedalam Netbeans.
- 22 5. Melakukan pengujian perangkat lunak dengan berbagai kemungkinan kasus.
- 23 6. Menyimpulkan atas serangkaian pengembangan yang telah dilakukan.

24 1.6 Sistematika Pembahasan

- 25 1. Bab 1 Pendahuluan
26 Bab ini berisi tentang latar belakang, rumusan masalah, tujuan, batasan masalah,
27 metodologi penelitian, dan sistematika pembahasan.
- 28 2. Bab 2 Dasar Teori
29 Bab ini berisi tentang teori dasar tentang Java FX, Apache POI, iCal4j, Konsep MVC,
30 dan Netbeans.
- 31 3. Bab 3 Analisis
32 Bab ini berisi tentang analisis kebutuhan dan fitur PL, diagram aktifitas PL, use case,
33 diagram kelas.
- 34 4. Bab 4 Perancangan
35 Bab ini berisi tentang perancangan kelas dalam PL dan gambaran *user interface*.
- 36 5. Bab 5 Implementasi dan Pengujian
37 Bab ini berisi tentang penerapan hasil rancangan pada bab sebelumnya serta pengujian
38 perangkat lunak.

¹ 6. Bab 6 Kesimpulan dan Saran

² Bab ini berisi tentang kesimpulan yang didapatkan dari hasil pengujian serta saran
³ apabila ingin melanjutkan pengembangan ini.

¹

BAB 2

²

DASAR TEORI

³ Pada bab ini akan dijelaskan mengenai konsep-konsep dasar pengukung BI, yaitu DSS(*decision*
⁴ *support system*), konsep BI(*business intelligence*) itu sendiri, *data warehouse* dengan OLAP(*online*
⁵ *analytical processing*), ORM(*object relational mapping*), MVC(*model view controller*), serta
⁶ penjelasan mengenai perangkat lunak OODOO yang digunakan untuk membangun BI *tool*.

⁷ 2.1 Decision Support Systems

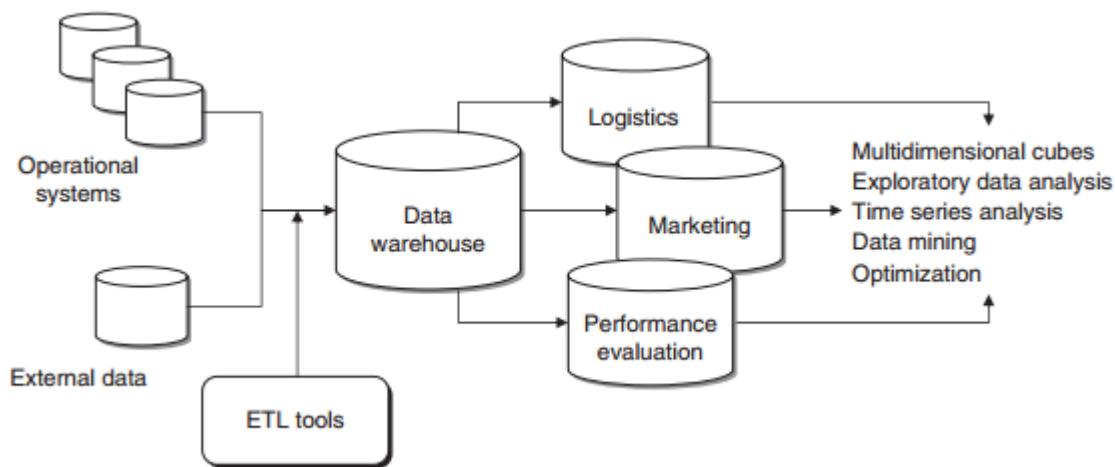
⁸ *Decision support systems* atau yang biasa disingkat DSS merupakan sebuah set pengembangan
⁹ teknik IT yang interaktif atau bisa dikatakan sebagai sebuah alat yang didesain untuk
¹⁰ menganalisis dan memproses data untuk membantu seorang manajer di suatu perusahaan
¹¹ dalam membuat keputusan. Dalam membuat sebuah keputusan, sistem dapat mencocokan
¹² sumber data individual dari seorang manajer dengan sumber data yang dimiliki oleh komputer
¹³ sehingga dapat meningkatkan kualitas keputusan yang akan dipilih [2]. Fungsi DSS
¹⁴ dapat dicerminkan sebagai konsep dari pengambilan keputusan dalam kondisi yang berbeda-
¹⁵ beda, akan sangat berguna untuk memecahkan masalah yang tidak terstruktur maupun semi
¹⁶ terstruktur. Dimana proses pengambilan keputusan terjadi dengan adanya dialog interaktif
¹⁷ antara DSS dan pengguna. Adanya *user interface* menandakan *user* tetap memegang
¹⁸ kontrol penuh dalam pengambilan keputusan [3].

¹⁹ 2.2 Business Intelligence

²⁰ *Business Intelligence* pada hakikatnya merupakan solusi perangkat lunak bagi *decision-makers*
²¹ dalam perusahaan yang dapat membantu *decision-makers* mengidentifikasi dan
²² memahami kunci dari faktor bisnis yang dijalani oleh perusahaan tersebut. BI juga dapat
²³ membantu *decision-makers* mengambil keputusan terbaik untuk situasi yang berlaku pada
²⁴ saat itu[2]. Pengertian lain *business intelligence* (BI) yaitu aplikasi *e-business* yang berfungsi
²⁵ untuk mengubah data dalam perusahaan (data operasional, transaksional, dan lainnya) ke
²⁶ dalam bentuk ringkasan informasi yang dibutuhkan. Tujuan utama dari sistem *business*
²⁷ *intelligence* adalah untuk menyediakan informasi untuk dapat membuat keputusan yang tepat
²⁸ dan efektif. Beberapa fungsi BI di antaranya menyajikan laporan(*query* dan *reporting*),
²⁹ eksplorasi data(*data mining*), analisis secara *online*(*online analytical processing* (OLAP)),
³⁰ pembandingan(*benchmarking*), dan pencarian teks(*text mining*). [1]

³¹ 2.2.1 Arsitektur business intelligence

³² Arsitektur sistem BI memiliki 3 komponen utama, yaitu *data sources*, *data Data warehouses*
³³ dan *data marts*. Penjelasan lebih lanjut dan bentuk arsitektur BI dapat dilihat pada Gambar
³⁴ 2.1.



Gambar 2.1: Komponen utama dari sistem *business intelligence* [1]

1 Data Sources

2 Pada arsitektur BI, tahap pertama yang dilakukan adalah pengumpulan dan pengintegrasian
 3 data dari berbagai sumber dengan tipe yang beragam. Sumber data biasanya berasal dari
 4 sistem operasional. Selain itu, sumber data dapat berupa dokumen yang tidak mempunyai
 5 struktur seperti email, teks atau data eksternal lainnya.[1]

6 Data Warehouses dan Data Marts

7 Setelah sumber data terkumpul, selanjutnya data akan masuk dalam proses ekstrasi dan
 8 transformasi yang biasa disebut dengan proses *extract, transform, load* (ETL). Data yang
 9 telah melalui proses ETL disimpan dalam basis data sebagai *data warehouses* dan *data*
 10 *marts*. [1]

11 Analisis BI

12 Data yang telah melalui proses ETL dan disimpan sebagai *data warehouse* kemudian di
 13 analisis oleh sistem BI. Beberapa contoh analisis BI, yaitu :

- 14 • analisis *multidimensional cube* merupakan analisis berbasis data array multidimensi
- 15 • analisis eksplorasi data merupakan analisis berupa statistik dan visual.
- 16 • analisis berdasarkan waktu merupakan analisis berbasis pada waktu tertentu.
- 17 • model pembelajaran secara induktif untuk *data mining* merupakan model untuk mem-
 18 pelajari apa yang ada pada data.
- 19 • model optimisasi merupakan optomisasi untuk memilih alternatif terbaik untuk disa-
 20 jikan sebagai bahan pertimbangan pengambilan keputusan.

21 2.3 Data Warehouse

22 Pada bagian ini akan dibahas mengenai pengertian dari *data warehouse*, karakteristik *data*
 23 *warehouse*, arsitektur *data warehouse* serta peran *data warehouse* dalam BI .

2.3.1 Definisi *Data Warehouse*

Sebelum membahas lebih jauh terdapat dua tipe data yang sering dipakai oleh perusahaan yaitu data operasional dan *data warehouse*. Terdapat perbedaan mendasar dari kedua data tersebut, data operasional biasanya mencakup periode waktu yang singkat, hal ini karena data operasional biasanya berisi data transaksi yang diperbarui setiap harinya. Sedangkan *data warehouse* mempunyai lingkup yang lebih luas yaitu menyimpan data yang mencakup beberapa tahun untuk proses analisis. Oleh karena itu, *data warehouse* diperbarui secara berkala dari data operasional agar selalu berkembang. Secara konsep *Data warehouse* (DWH) merupakan kumpulan teknik, metoda atau bisa dikatakan sebagai suatu *tools* yang dapat membantu manajer, direktur maupun analis perusahaan untuk melakukan analisis data sebelum mengambil keputusan. Secara singkat *data warehouse* dapat di artikan sebagai sekumpulan data yang membantu proses pengambilan keputusan. Karakteristik DWH yaitu *subject-oriented*, terintegrasi, konsisten, berevolusi dari waktu ke waktu, dan tidak berubah-ubah(*not volatile*) [2].

Characteristic	OLTP	OLAP
volatility	dynamic data	static data
timeliness	current data only	current and historical data
time dimension	implicit and current	explicit and variant
granularity	detailed data	aggregated and consolidated data
updating activities	continuous and irregular repetitive	periodic and regular unpredictable
flexibility	low	high
performance	high, few seconds per query	may be low for complex queries
users	employees	knowledge workers
functions	operational	analytical
purpose of use	transactions	complex queries and decision support
priority	high performance	high flexibility
metrics	transaction rate	effective response
size	megabytes to gigabytes	gigabytes to terabytes

Gambar 2.2: Perbedaan OLTP dan OLAP sistem [1]

Desain *data warehouse* memiliki perbedaan mendasar dengan *operational databases*. Gambar 2.2 merangkum perbedaan mendasar antara OLTP(*on-line transaction processing*) dan OLAP(*on-line analytical processing*) sistem.

Karakteristik *Data Warehouse*

Data Warehouse memiliki empat karakteristik utama, yaitu [2].

1. *Subject Oriented*
Data warehouse dirancang, dan dibangun untuk memenuhi kebutuhan analisis berdasarkan subyek tertentu. Misalnya, *data warehouse* untuk *customer* atau produk.
2. *Integrated*
Data warehouse mampu mengintegrasikan data sumber yang berbeda-beda kedalam format yang seragam.
3. *Non Volatile*
Format data dalam sebuah *data warehouse* tidak bisa diubah karena merupakan data historis, data dalam *data warehouse* hanya boleh ditambah saja.
4. *Time Variant*
Setiap data yang disimpan dalam *data warehouse* memiliki dimensi waktu. Dimensi waktu tersebut selanjutnya akan dimanfaatkan sebagai pembanding dalam pembuatan laporan.

1 Data Marts

2 *Data marts* merupakan bagian dari kumpulan data yang disimpan dalam *data warehouse*. Kumpulan data tersebut mengandung satu set bagian informasi yang relevan dan merujuk ke arah area bisnis yang spesifik, seperti bagian departemen dalam suatu perusahaan atau kategori pengguna. Jadi cakupan *data mart* lebih spesifik dibanding *data warehouse* [2].
3 Ada beberapa karakteristik dari data mart yang membedakannya dengan data warehouse,
4 yaitu.[1]

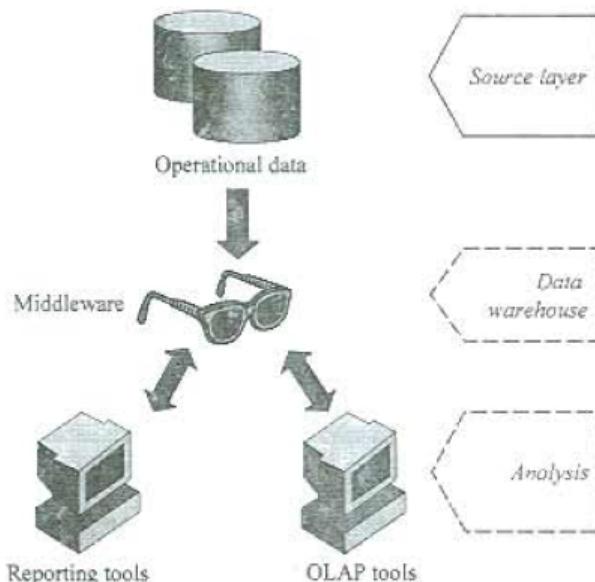
- 5 1. Data mart lebih fokus pada kebutuhan pengguna, departemen atau fungsi bisnis.
- 6 2. Data mart tidak berisi data operasional yang terperinci.
- 7 3. Data mart berisi lebih sedikit data dari data warehouse, sehingga lebih mudah dimengerti dan dipahami.

12 2.3.2 Arsitektur *Data Warehouse*

13 Arsitektur *data warehouse* dapat dikelompokkan menjadi tiga, yaitu[2]:

14 Single-Layer Architecture

15 *Single-layer architecture* bertujuan untuk meminimalisir jumlah data yang disimpan dengan
16 cara mengilangkan redundansi data.

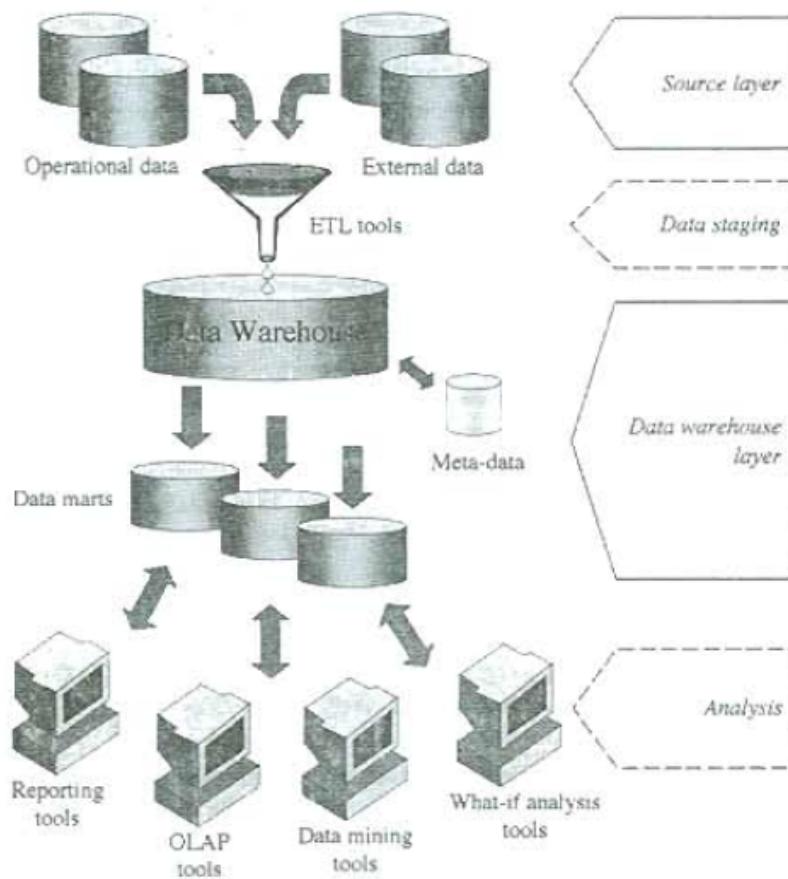


Gambar 2.3: Gambar *Single-Layer Architecture*[2]

17 Diterangkan pada gambar 2.3 bahwa hanya terdapat satu layer dalam arsitektur ini, ya-
18 itu *source layer* yang berisi data operasional. *Data warehouse* dianggap sebagai *virtual layer*
19 sehingga *data warehouse* dalam konteks ini dianggap sebagai tampilan multidimensional dari
20 data operasional yang secara spesifik disebut *middleware* atau *intermediate processing layer*.
21 Kekurangan arsitektur ini terletak pada sulitnya menemukan prasyarat untuk memisahkan
22 proses analisis dan transaksi.[2]

23 Two-Layer Architecture

24 Arsitektur ini menggunakan dua layer yaitu memisahkan antara sumber data fisik dan *data*
25 *warehouse*. Terdapat empat bagian sebagai pendukung aliran data pada arsitektur ini,
26 yaitu:



Gambar 2.4: Gambar Two-Layer Architecture[2]

1 1. *Source layer*

2 Sebuah *data warehouse* terdiri dari sumber data yang heterogen yang berasal dari
3 data operasional perusahaan, basis data warisan, maupun data yang berasal dari luar
4 lingkup perusahaan.

5 2. *Data staging*

6 *Data staging* merupakan tempat penyimpanan sementara untuk proses ETL(*extract, transform, load*).

8 3. *Data warehouse layer*

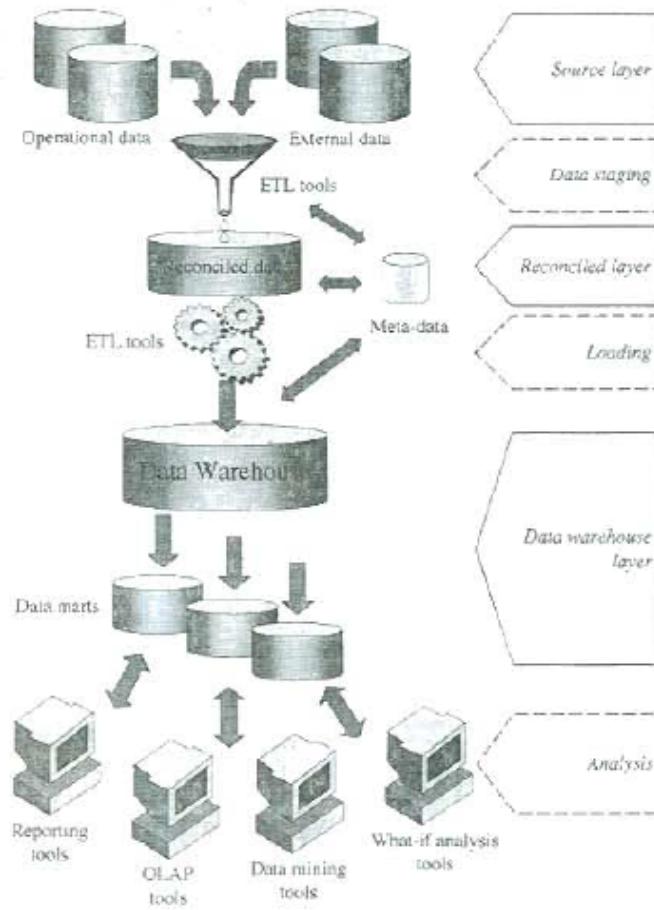
9 Setelah data melalui proses ETL, data akan disimpan dalam satu penyimpanan tunggal
10 yang tersentralisasi yang disebut *data warehouse*. *Data warehouse* bisa digunakan
11 sebagai sumber untuk membuat *data mart* untuk kebutuhan yang lebih spesifik.

12 4. Analisis

13 Dalam layer ini, data yang terintegrasi bersifat efisien dan fleksibel untuk diakses
14 sehingga dapat digunakan untuk membuat laporan, melakukan analisis data, dan lain-
15 lain.

16 **Three-Layer Architecture**

17 Dalam arsitektur ini, terdapat layer ketiga yang disebut dengan *reconciled data layer*. La-
18 yer ketiga ini tercapai ketika mendapatkan data operasional yang terintegrasi dan telah
19 dibersihkan dari data sumbernya. Isi dari *reconciled data layer* ini adalah data yang telah
20 terintegrasi, konsisten, data yang benar, data terkini dan detil. Keuntungan dari peng-
21 gunaan *reconciled data layer* ini adalah terciptanya suatu model data yang dapat menjadi



Gambar 2.5: Gambar *Three-Layer Architecture*[2]

- 1 referensi bagi seluruh data perusahaan. Kekurangannya adalah penggunaan *reconciled data*
 2 *layer* dapat mengarah pada redundansi pada sumber data operasional[2].

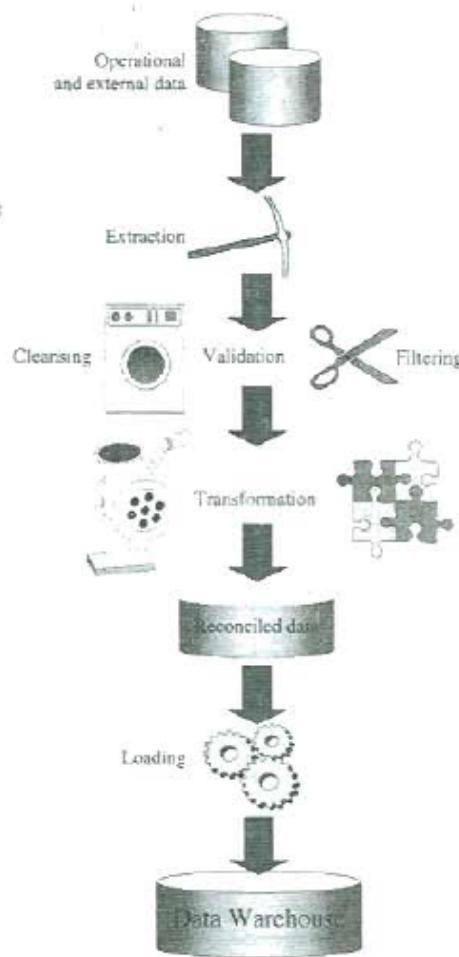
3 2.3.3 Proses ETCL

- 4 Proses mentransformasikan data sumber kedalam *data warehouse* dengan berbagai macam
 5 variasi data pada sumber disebut proses ETCL. Proses ETCL terdiri dari empat tahap,
 6 yaitu *extract*, *transform*, *clean*, dan *load*. Proses ETCL merupakan proses utama dalam
 7 membangun *data warehouse* dan *data mart*. Berikut penjelasan mengenai tahapan dari
 8 ETCL dengan disertai gambar 2.6.

9 *Extract*

- 10 *Data warehouse* terdiri dari berbagai sumber data. Proses ekstraksi pada ETCL berfungsi
 11 untuk mendapatkan data yang relevan sesuai dengan kebutuhan *data warehouse*. Proses
 12 ekstraksi terbagi menjadi dua tipe, yaitu [2]:

- 13 1. *Static extraction*
 14 *Static extraction* adalah proses pengisian sebuah *data warehouse* untuk pertama kali.
 15 Proses ini seperti gambaran dari data operasional.
- 16 2. *Incremental extraction*
 17 *Incremental extraction* digunakan untuk proses memperbarui *data warehouse* secara
 18 berkala. Proses ini juga untuk mengaplikasikan perubahan dari sumber data sejak
 19 terakhir kali proses ekstraksi.



Gambar 2.6: Contoh Proses ETCL Pada *Data Warehouses*[2]

1 *Cleansing*

2 Fase membersihkan data merupakan fase penting dalam sistem *data warehouse* karena dapat
 3 meningkatkan kualitas data karena pada umumnya sumber data tidak memiliki kualitas yang
 4 memadai untuk dimasukan dalam *data warehouse*. Berikut ini merupakan kesalahan yang
 5 sering dilakukan serta inkonsistensi data pada sumber data[2]:

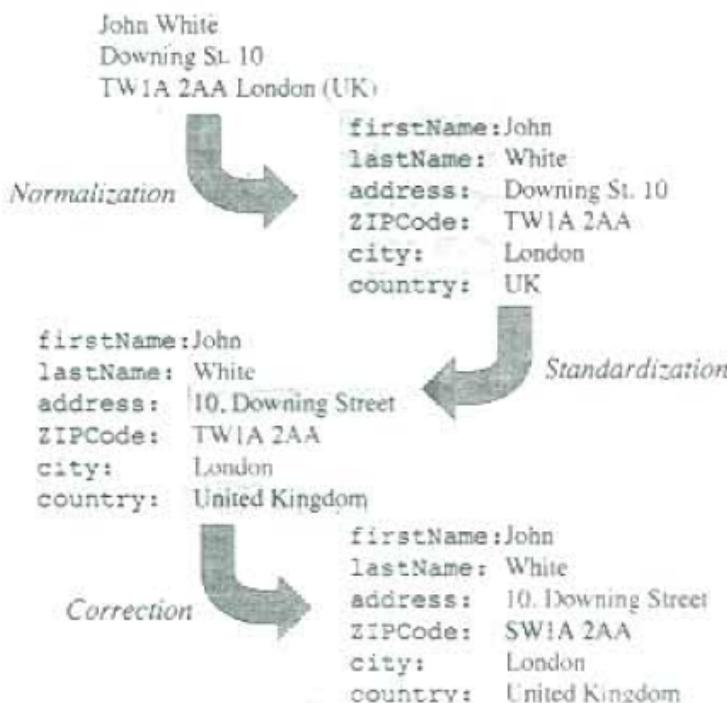
- 6 1. Duplikasi data
 7 Contohnya : Seorang pelanggan yang tercatat berulang kali dalam sistem manajemen
 8 suatu perusahaan.
- 9 2. Nilai-nilai yang tidak konsisten namun secara logika nilai tersebut saling berasosiasi
 10 Contohnya : alamat dan kode pos
- 11 3. Data yang hilang
 12 Contohnya : Pekerjaan pelanggan yang sering dikosongkan.
- 13 4. Penempatan nilai kolom yang salah
 14 Contohnya: kolom NIK Ktp diisi dengan nomer telepon
- 15 5. Nilai yang tidak mungkin salah
 16 Contohnya : Tanggal dan waktu hari ini.
- 17 6. Nilai yang tidak konsisten pada penggunaan singkatan.
 18 Contohnya : Penggunaan singkatan I untuk italy dan IT untuk nilai yang sama.

- 1 7. Nilai yang tidak konsisten akibat kesalahan pengetikan
 2 Contohnya : Jalan dago menjadi jalan dagu.

3 ***Transform***

4 Transformasi merupakan inti dari fase rekonsiliasi pada *data warehouse*. Tahap transformasi
 5 pada DWH adalah mengubah data dari sumber data operasional menjadi sebuah format
 6 yang spesifik pada *data warehouse*. Jika mengimplementasikan arsitektur *three-layer* maka
 7 *output* dari tahap ini akan masuk kedalam *reconciled data layer*. Berikut ini adalah hal
 8 utama yang dilakukan dalam proses transformasi [2]:

- 9 1. konversi dan normalisasi untuk menciptakan keseragaman data
 10 2. mencocokan *field-field* yang ekivalen dari sumber data yang berbeda.
 11 3. memilih *field* dan *record* untuk mengurangi jumlah data.



Gambar 2.7: Contoh Proses Transformasi Pada *Data Warehouses* [2]

12 Gambar 2.7 merupakan contoh dari proses *cleansing* dan transformasi dari data pelanggan
 13 dimana data berbasis kolom yang terstruktur diekstrak dari sebuah teks yang tidak ters-
 14 truktur, dengan beberapa nilai kolom distandarisasi sehingga menghilangkan singkatan, dan
 15 bahkan nilai dan kolom tersebut saling berkaitan sehingga memudahkan untuk diperbaiki [2].

16 ***Loading***

17 *Loading* adalah tahap terakhir dari proses ETCL. Kegiatan ini dapat dilakukan dengan dua
 18 cara [2]:

- 19 1. *Refresh*
 20 Data dalam *data warehouse* seluruhnya ditulis ulang, hal ini menandakan data awal
 21 diganti dengan data baru. Proses *refresh* biasanya dikombinasikan dengan *static extraction*
 22 ketika pertama membangun *data warehouse*.

1 2. *Update*

2 Data yang dimasukan ke dalam *data warehouse* hanya data pada sumber data yang
 3 mengalami perubahan atau penambahan saja. *Update* biasanya dilakukan tanpa meng-
 4 hapus atau mengubah data yang ada. Proses *update* biasanya dikombinasikan dengan
 5 *incremental extraction* untuk melakukan pembaharuan data secara berkala.

6 **2.3.4 Struktur *Data Warehouse* Berdasarkan Konsep Data Multidimensi**

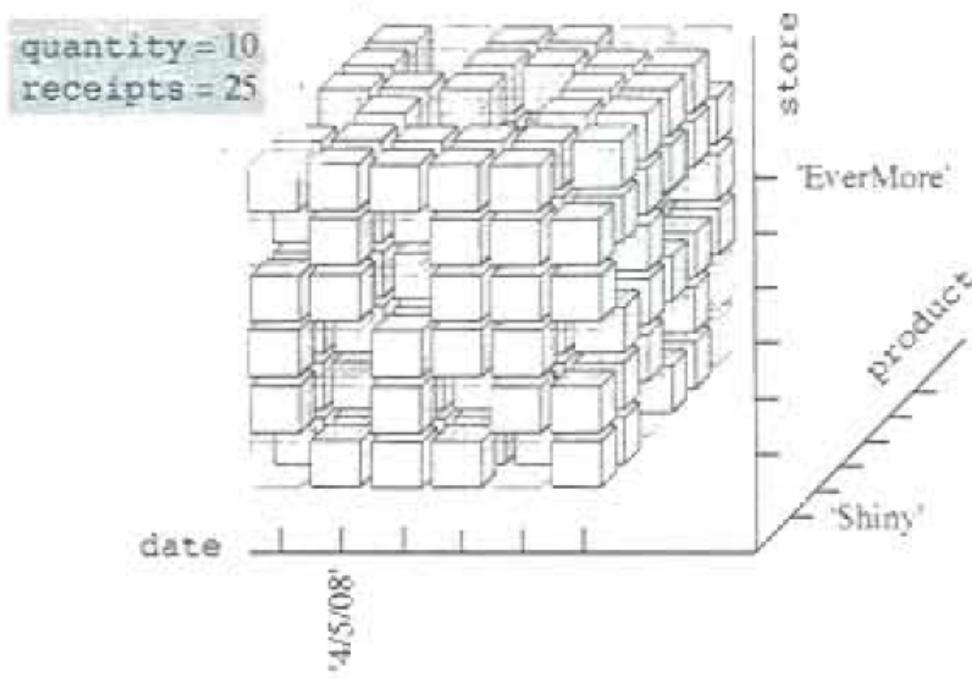
7 *Data warehouse* merupakan sumber data utama yang akan dianalisis dan diolah menggunakan teknik OLAP. OLAP(*online analytical processing*) menggunakan konsep data multidimensi(terdiri dari banyak dimensi) sehingga struktur *data warehouse* menyerupai kubus (*cube*). Struktur *cube* terdiri dari tiga komponen utama, yaitu *fact*, *dimension*, dan *measure*.
 11 Berikut ini penjelasan mengenai komponen *cube* disertai contoh gambar.[2]

12 1. *Fact*

13 *Fact* merupakan kumpulan peristiwa (*event*) dalam suatu perusahaan. Contohnya adalah *fact sales* untuk kegiatan penjualan dan *fact shipment* untuk kegiatan pengiriman.

16 2. *Dimension* merupakan bagian dari *fact*, namun *dimension* mempunyai lingkup lebih spesifik dari *fact*. Contoh: untuk *fact sales*, terdapat dimensi waktu, produk serta jenis barang.

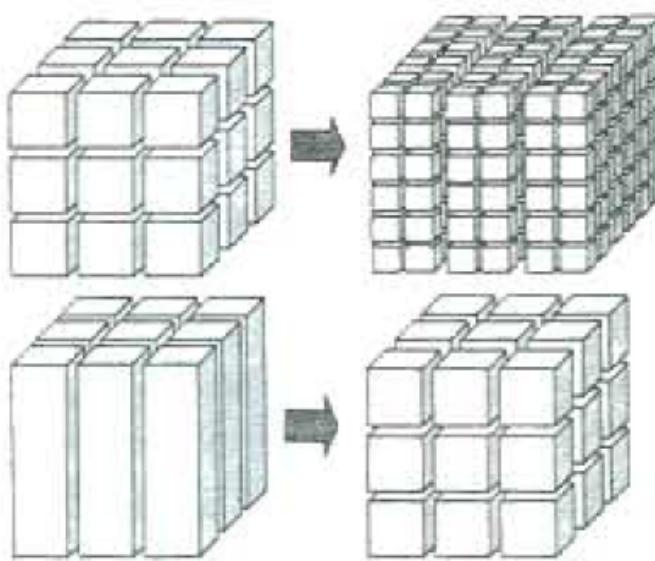
19 3. *Measure*, Merupakan bagian dari *fact*, dimana *measure* biasanya mendeskripsikan *fact* dengan angka atau numerik. Contohnya: *quantity* (banyaknya barang), *unitPrice* (harga per unit), *numberOfCustomer* (banyaknya pelanggan).



Gambar 2.8: Contoh *Three Dimensional Cube*[2]

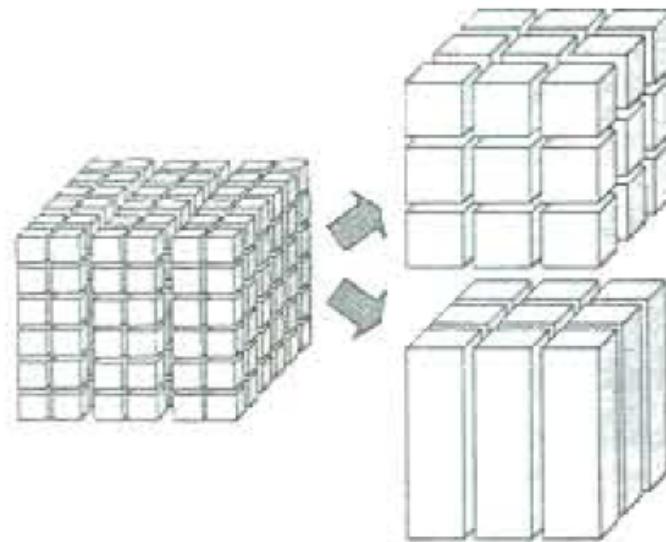
22 Gambar 2.8 menggambarkan sebuah *fact sales* memiliki tiga *dimension*, yaitu *date*, *store*, dan *product*. Serta memiliki dua *measure*, yaitu *quantity* dan *receipt*. *Cube* tersebut menginformasikan bahwa di toko *EverMore* telah terjual 10 product *Shing* dengan total harga 25 dollar pada tanggal 4 Mei 2008.[2]

- ¹ OLAP mempunyai teknik lain untuk mengolah *cube*, seperti *roll-up*, *drill-down* dan *slice-dice*.



Gambar 2.9: *Drill-Down*[2]

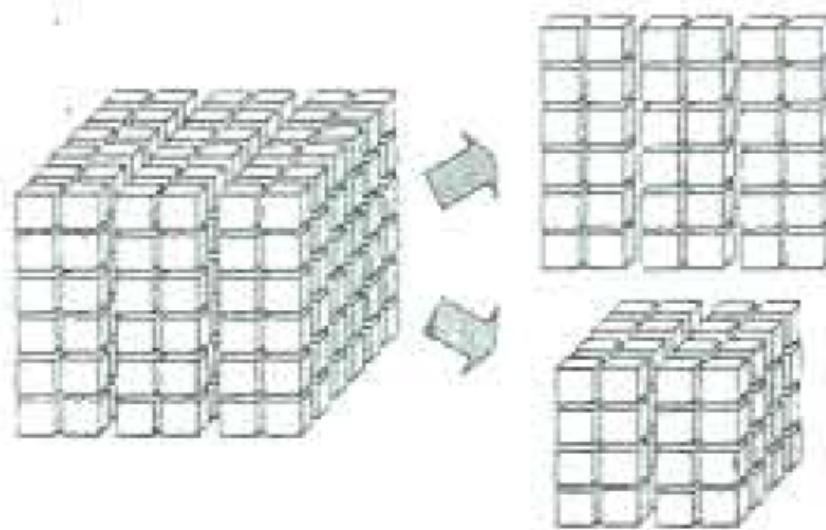
- ³ Gambar 2.9 menggambarkan proses *drill-down*. *Drill-down* sendiri merupakan kebalikan
⁴ dari *roll-up*. Pada gambar 2.9 teknik *drill-down* digunakan untuk mengurangi agregasi data
⁵ dan melihat suatu informasi secara lebih detil.



Gambar 2.10: *Roll-Up*[2]

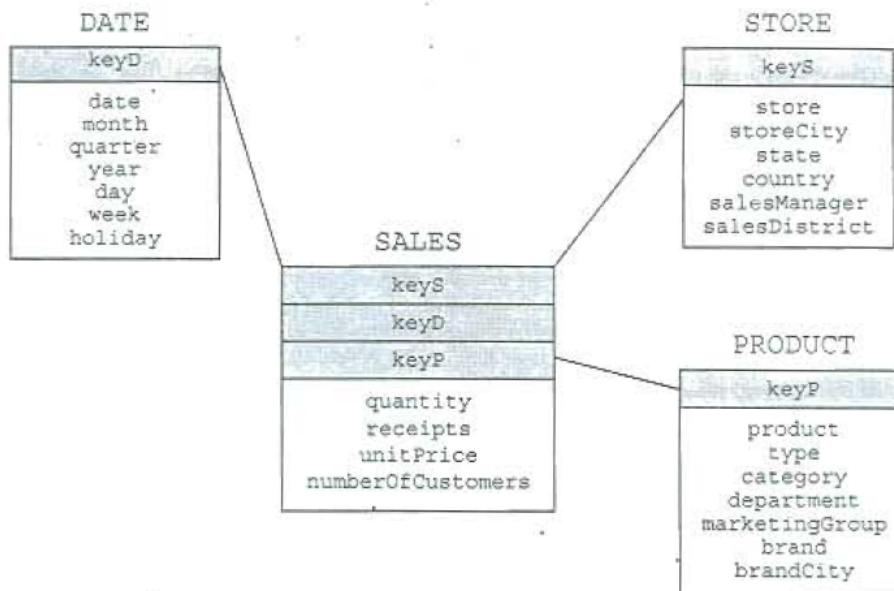
- ⁶ Gambar 2.10 menunjukkan bahwa *roll-up* digunakan untuk melihat informasi secara
⁷ umum(*general*).

- ⁸ Sedangkan untuk *slice-dice* dapat dilihat pada gambar 2.11 bahwa *slice* merupakan
⁹ proses mengurangi *dimension* pada *cube* setelah menempatkan salah satu *dimension* lain
¹⁰ menjadi lebih spesifik. *Dice* merupakan proses mengurangi sejumlah data untuk dianalisis
¹¹ dengan cara menyeleksi kriteria tertentu.

Gambar 2.11: *Slice*(Atas) dan *Dice*(Bawah)[2]

¹ 2.3.5 Teknik Pemodelan Dimensional

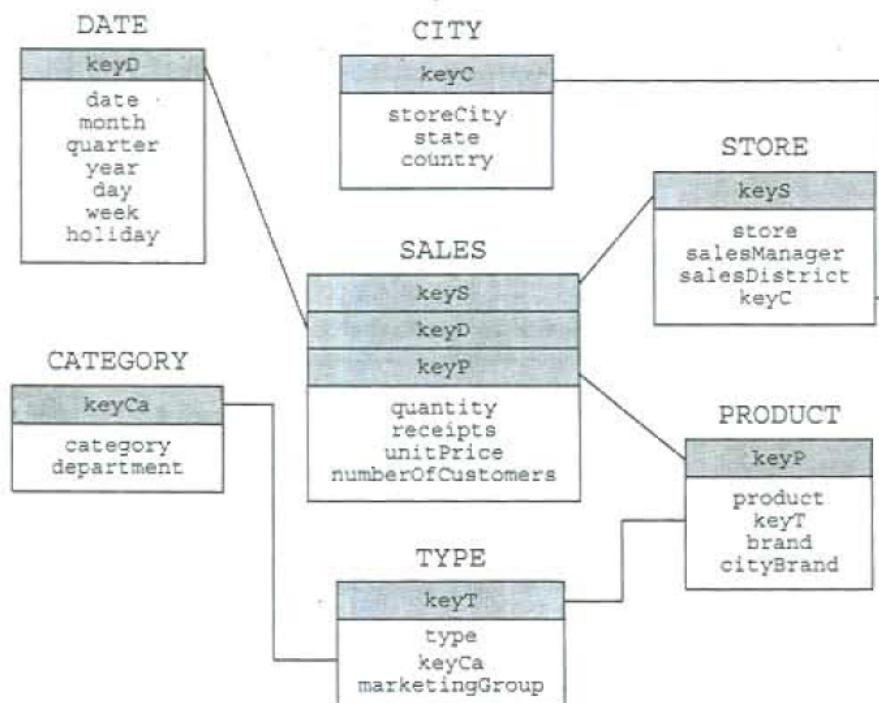
² Teknik pemodelan dimensional digunakan untuk menampilkan data dalam *data warehouse*, memungkinkan pengaksesan basis data dengan performansi lebih tinggi, dan membuat
³ struktur basis data menjadi konsisten. *Fact table* berisi beberapa *foreign key* yang berasal
⁴ dari *dimension table* dan atribut yang berasal dari hasil kalkulasi basis data operasional.
⁵ *Fact table* berisikan data-data historis, sedangkan *dimension table* memiliki informasi yang
⁶ mendukung *fact table* dengan format yang lebih spesifik namun masih memiliki *primary key*
⁷ yang sesuai dengan *fact table*. Pemodelan ini umumnya dijadikan sebagai pendekatan untuk
⁸ menggambarkan situasi bisnis. Secara umum, terdapat tiga cara pemodelan dimensional,
⁹ yaitu *star schema*, *snowflake schema*, dan *constellation schema* [2].

Gambar 2.12: *Star Schema*[2]

Skema ini terdiri dari sebuah *fact table* yang berhubungan langsung dengan beberapa *dimension table*. Biasanya letak *fact table* berada di tengah dan dikelilingi *dimension table* yang menyerupai bentuk bintang. Keuntungan dari *star schema* ini adalah lebih sederhana sehingga mudah dipahami oleh pengguna. Selain itu, navigasi melalui basis data yang memakai *star schema* lebih optimal dan lebih mudah, meskipun hasil *query* terlihat kompleks.[2]

2. Snowflake Schema

Skema ini adalah variasi bentuk dari *star schema*. Letak perbedaannya ada pada *dimension table*. *Dimension table* pada *star schema* dinormalisasi (memperbaiki desain tabel sehingga penyimpanan data jauh lebih efisien dan bebas anomali data) sehingga menghasilkan tabel baru, namun tetap berhubungan dengan *dimension table* yang bersangkutan. Keuntungan dari skema ini adalah menghemat memori, mengurangi terjadinya redundansi, dan strukturnya yang normal memudahkan proses *update*. Kelemahannya ada pada skema yang dihasilkan kompleks sehingga sulit melakukan pencarian terhadap isi skema. Berikut contoh dari skema *snowflakes star schema*.[2]



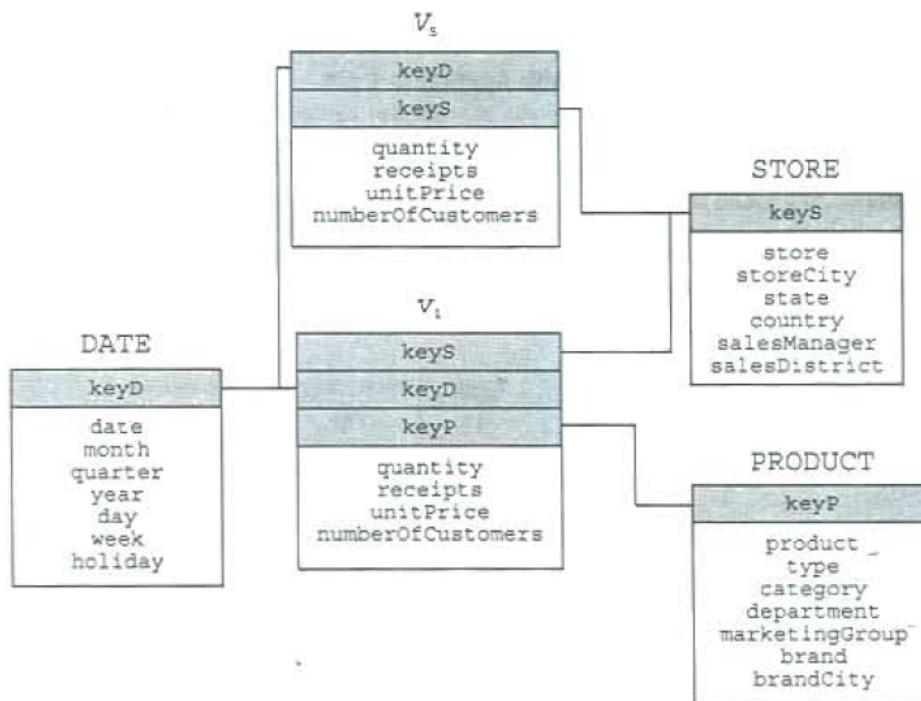
Gambar 2.13: *Snowflakes Schema*[2]

3. Constellation schema

Constellation schema merupakan gabungan beberapa *star schema*[2]. Model ini terdiri dari beberapa *fact table* dan saling terbubung dengan *dimension table* bersangkutan. Jadi, sebuah *dimension table* dapat berhubungan dengan dua skema bahkan lebih sehingga menghemat memori dan lebih efektif. Namun, model ini tidak cocok untuk jumlah data yang sangat besar dan kompleks karena memungkinkan proses kalkulasu dari aggregasi data akan memakan waktu yang relatif lama. Berikut contoh skema *Constellation*.

2.3.6 Hubungan OLAP, Data warehouse, DSS, dengan Business Intelligence

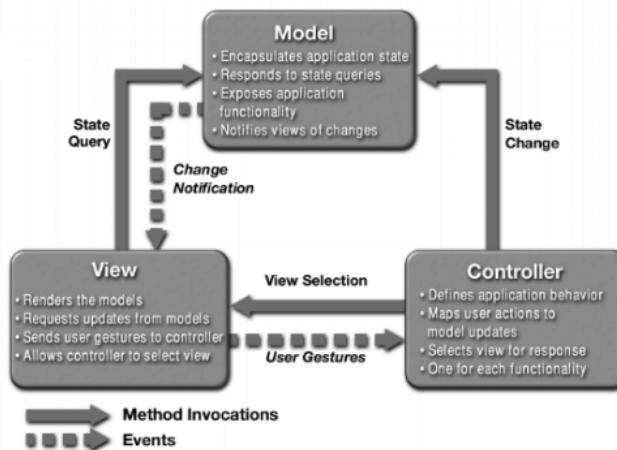
Data warehouse akan diolah dengan teknik OLAP agar dapat di analisis dan dievaluasi secara cepat dan efisien. Teknologi OLAP tersebut disediakan oleh BI. Hasil pengolahan *data*

Gambar 2.14: *Constellation Schema*[2]

- ¹ warehouse akan dimanfaatkan dalam DSS untuk mengambil keputusan-keputusan penting
² terkait kepentingan perusahaan. [2]

³ 2.4 Model View Controller(MVC)

- ⁴ Setiap perangkat lunak memiliki pola bawaan dalam proses pembuatannya. Salah satu
⁵ pola yang cukup terkenal adalah konsep MVC. MVC memisahkan pengembangan aplikasi
⁶ berdasarkan komponen utama yang membangun sebuah aplikasi seperti manipulasi data, user interface, dan bagian yang menjadi kontrol aplikasi.



Gambar 2.15: Diagram MVC

- ⁷
⁸ Penjelasan MVC sebagai berikut [4]:
- ⁹ • *Model*
 - ¹⁰ Komponen ini berhubungan dengan proses bisnis yang ada dan juga terdapat di dalam sistem.
 - ¹¹ Contoh model di OODOO adalah tabel-tabel yang berada di postgresql.

1 ● *Controller*

2 Komponen yang mengatur komponen dari model ke view maupun sebaliknya. Fungsi
3 dari komponen ini adalah sebagai penghubung komponen model dan view.

4 Contoh *Controller* di OODOO adalah kelas-kelas yang diinisiasi menjadi objek. Setiap
5 kelas yang dibuat harus meng-*extend* kelas osv yang merupakan syarat dari *framework*
6 tersebut.

7 ● *View*

8 komponen yang berhubungan dengan *end user*. Bentuknya berupa tampilan yang akan
9 digunakan oleh *user* atau sering disebut sebagai *user-interface*. Contoh *view* di OODOO
10 adalah kode-kode yang dituliskan dalam *file* yang berekstensi xml.

11 Banyak keuntungan dengan pendekatan MVC, yakni [4]:

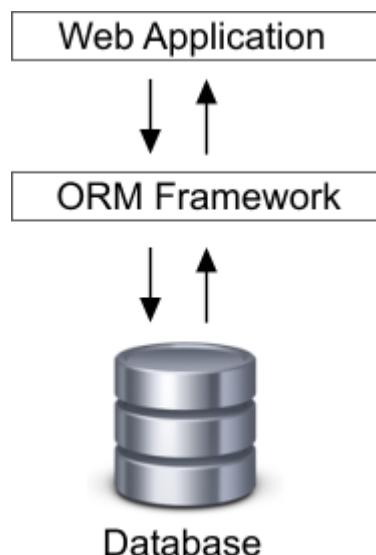
- 12 1. Pembagian modul pengerjaan yang jelas untuk setiap orang di dalam suatu tim.
- 13 2. Kemudahan untuk *maintenance* dimasa depan.
- 14 3. Mempunyai fleksibilitas tinggi dalam pengembangan,

15 1

16 **2.5 Object Relational Mapping(ORM)**

17 ORM merupakan teknik *programming* yang memetakan basis data relasional ke model ob-
18 jek. Model object dituliskan dalam bentuk bahasa *object-oriented programming* seperti java,
19 C atau python. Pada dasarnya setiap database yang tersimpan dalam bentuk byte atau
20 array. ORM merubah data di antara berbagai tipe sistem berbeda kedalam database rela-
21 sional dan OOP. [4]

22 Dalam bahasa pemograman yang menggunakan ORM data-data yang tersimpan dalam da-
23 tabase aplikasi seperti MySQL atau PostgreSQL, dipetakan menjadi objek sebagai penghu-
24 bung antara objek yang dibangun dalam program dengan database², Seperti terlihat pada
gambar 2.16 Beberapa keuntungan menggunakan ORM, diantaranya: ³



Gambar 2.16: Konsep ORM

25

¹<http://elmolya.blogspot.co.id/2010/03/belajar-mvc.html>

²<http://www.techopedia.com/definition/24200/object-relational-mapping-orm>

³<http://www.techopedia.com/definition/24200/object-relational-mapping-orm>

1. Pengembangan menjadi lebih sederhana karena ORM mengotomatisasi konversi objek kedalam tabel maupun sebaliknya. Hasil dalam pengembangan ini adalah biaya pemeliharaan yang lebih rendah.
2. Mengurangi penulisan code untuk pemanggilan SQL dan prosedur penyimpanan data dalam database.
3. Meningkatkan performansi sistem.
4. Solusi optimal yang dapat membuat sistem menjadi lebih cepat dan mudah dalam pemeliharaan sistem.

2.6 Framework OODOO

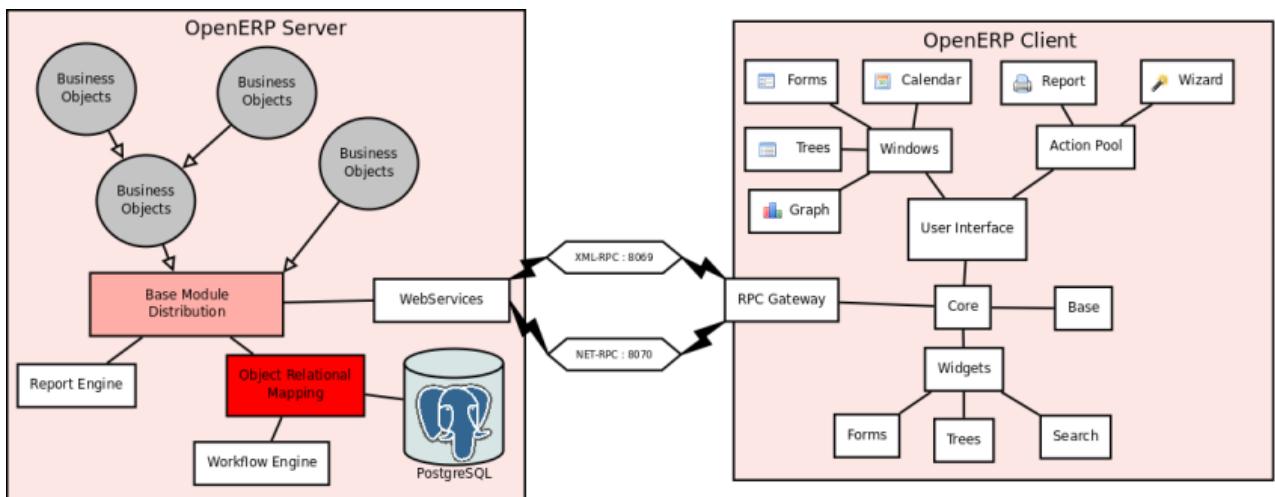
ODOO adalah aplikasi open source guna memanajemen ERP sebuah usaha atau lebih tepatnya ERP sebuah perusahaan. OODOO memiliki tampilan yang cukup minimalis. Didalamnya tercantum manajemen penjualan, logistik, akutansi, pemberdayaan pekerja, dan produksi. Dengan sebuah aplikasi OODOO diharap ERP sebuah perusahaan dapat dijalankan semaksimum mungkin tanpa mengurangi unsur-unsur lain dalam usaha didalamnya.⁴

2.6.1 Arsitektur OODOO

Bagian ini menyajikan arsitektur OODOO bersama dengan rincian teknologi aplikasi. Disajikan tingkatan penyusunan OODOO, sarana komunikasi dan protokol antara komponen aplikasi. Seperti terlihat pada gambar 2.17

Client-Server Architecture

ODOO memiliki komponen *client server* terpisah, dengan kata lain Server berjalan secara terpisah dari klien. Server menangani logika bisnis dan berkomunikasi dengan aplikasi database, sedangkan klien menyajikan informasi kepada pengguna dan memungkinkan pengguna untuk beroperasi dengan aplikasi server. Disisi lain OODOO menyediakan multiple klien.



Gambar 2.17: Arsitektur OODOO

2.6.2 Server dan Modul

Server OODOO ditulis dalam bahasa pemrograman Python. Klien berkomunikasi dengan server dengan menggunakan antarmuka XML-RPC, sedangkan fungsi bisnis diatur dalam

⁴<https://doc.odoo.com/>

1 modul. Modul adalah folder dengan yang telah ditentukan struktur yang berisi kode dan
 2 XML *Python file*. Sebuah modul mendefinisikan struktur data, *form*, laporan, menu, pro-
 3 sedur, dan alur kerja. Modul didefinisikan menggunakan sintaks *client-independent*. Jadi,
 4 menambahkan objek baru, seperti menu atau bentuk, serta membuatnya tersedia untuk
 5 klien.⁵

6 Aplikasi Klien

7 Dua aplikasi Client adalah:⁶

- 8 1. Sebuah Aplikasi web yang digunakan sebagai HTTPserver memungkinkan pengguna
 9 untuk terhubung menggunakan web browser.
- 10 2. Desktop aplikasi yang ditulis dalam Python yang menggunakan GTK (*Graphics tool
 11 Kit*)

12 Database

13 OODO menggunakan PostgreSQL sebagai sistem manajemen database. PostgreSQL adalah
 14 sistem manajemen database objek-relasional (ORDBMS)

15 *Report*:⁷

16 OODO juga menyediakan sistem pelaporan dengan integrasi OpenOffice.org memungkinkan
 17 kustomisasi laporan. OpenOffice.org dengan komponen utamanya adalah untuk pengolah
 18 kata, *spreadsheet*, presentasi, grafis, dan *database*.

19 Tiga Tingkatan Arsitektur DOO

20 OODO adalah *multitenant* dengan tiga tingkatan arsitektur : *Database tier* untuk penyim-
 21 panan data, *application tier* untuk pengolahan dan *functionalities* dan *presentation tier* me-
 22 nyediakan antarmuka pengguna. *Database*, *application*, *functionalities*, dan *presentation*
 23 adalah lapisan yang terpisah dalam OODO. *Application tier* sendiri ditulis sebagai inti; be-
 24 berapa modul tambahan dapat diinstal dalam rangka menciptakan spesifikasi khusus dari
 25 OODO yang disesuaikan dengan kebutuhan dan persyaratan khusus. Selain itu, OODO
 26 mengadopsi pola arsitektur Model-View-Controller (MVC).⁸

27 Untuk mengakses OODO dapat dilakukan dengan cara:⁹

- 28 1. menggunakan browser web menunjuk klien-server web OODO, atau
- 29 2. menggunakan aplikasi client (klien GTK) diinstal pada setiap komputer.

30 Dua metode akses memberikan fasilitas yang sangat mirip, dan dapat menggunakan kedua
 31 akses server yang sama pada waktu yang sama. Cara terbaik adalah dengan menggunakan
 32 web browser jika server OODO adalah agak jauh (seperti di benua lain) karena lebih toleran
 33 terhadap penundaan waktu antara dua dari klien GTK adalah. Klien web juga lebih mu-
 34 dah untuk mempertahankan, karena umumnya sudah diinstal pada komputer pengguna.¹⁰
 35 Sebaliknya akan lebih baik dengan klien aplikasi (disebut klien GTK karena teknologi itu
 36 dibangun dengan) menggunakan server lokal (seperti di gedung yang sama). Dalam hal ini
 37 klien GTK akan lebih responsif, sehingga lebih memuaskan untuk digunakan.¹¹

⁵<https://doc.odoo.com/>

⁶<https://doc.odoo.com/>

⁷<https://doc.odoo.com/>

⁸<https://doc.odoo.com/>

⁹<https://doc.odoo.com/>

¹⁰<https://doc.odoo.com/>

¹¹<https://doc.odoo.com/>

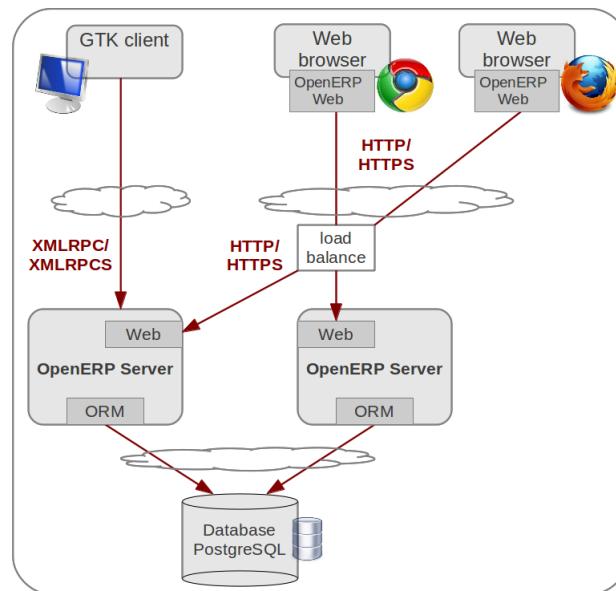
Ada sedikit perbedaan fungsional antara dua klien OODOO - klien web dan klien GTK saat ini. Klien web menawarkan fungsionalitas yang lebih, misalnya, fitur Intelijen Perusahaan, dan tampilan Gantt. Ketika mengubah struktur instalasi OODOO (menambah dan menghapus modul, mungkin mengubah label), mungkin menemukan klien web menjadi tidak tepat karena penggunaannya *caching*.¹²

Caching mempercepat semuanya dengan menjaga salinan data di suatu tempat antara server dan klien, yang biasanya baik. Tapi mungkin telah membuat perubahan pada instalasi bahwa tidak dapat langsung melihat di browser. Banyak kesalahan yang jelas disebabkan oleh perubahan struktur instalasi OODOO, Solusinya adalah dengan menggunakan klien GTK selama pengembangan dan implementasi di mungkinkan.

Perusahaan OODOO akan terus mendukung dua klien, yaitu *web browser*, GTK di masa mendatang, sehingga dapat menggunakan klien yang di inginkan.¹³

Sebuah penyebaran khas OODOO ditampilkan pada Gambar 2.18 . Penyebaran ini disebut Web tertanam penyebaran. Seperti ditunjukkan, sistem OODOO terdiri dari tiga komponen utama:

1. server PostgreSQL database, yang berisi semua database, yang masing-masing berisi semua data dan sebagian besar elemen konfigurasi sistem OODOO.
2. server aplikasi OODOO, yang berisi semua logika perusahaan dan memastikan bahwa OODOO berjalan secara optimal.
3. server web, aplikasi terpisah yang disebut Open Object client-web, yang memungkinkan untuk terhubung ke OODOO dari browser web standar dan tidak diperlukan saat menghubungkan menggunakan klien GTK.



Gambar 2.18: Arsitektur OODOO

23 Database PostgreSQL

Tierdata OODOO disediakan oleh database relasional PostgreSQL. Sementara query SQL langsung dapat dijalankan dari modul OODOO, sebagian mengakses ke database relasional dilakukan melalui server Object lapisan Pemetaan Relasional. Database berisi semua data aplikasi, dan juga sebagian besar elemen konfigurasi sistem OODOO. Perhatikan bahwa server ini mungkin dapat digunakan menggunakan database berkerumun.¹⁴

¹²<https://doc.odoo.com/>

¹³<https://doc.odoo.com/>

¹⁴<https://doc.odoo.com/>

¹ Server ODOO

² OODO menyediakan server aplikasi yang aplikasi bisnis tertentu dapat dibangun. Ini juga
³ merupakan kerangka pengembangan yang lengkap, menawarkan berbagai fitur untuk menu-
⁴ lis aplikasi mereka. Di antara fitur tersebut, OODO ORM menyediakan fungsionalitas dan
⁵ antarmuka di atas server PostgreSQL. Server OODO juga dilengkapi dengan lapisan khu-
⁶ sus yang dirancang untuk berkomunikasi dengan klien berbasis browser web. Lapisan ini
⁷ menghubungkan pengguna dengan menggunakan *browser standard* untuk server. Dari pers-
⁸ pektif pengembang, server bertindak baik sebagai perpustakaan yang membawa manfaat di
⁹ atas sementara menyembunyikan rincian tingkat rendah, dan sebagai cara sederhana untuk
¹⁰ menginstal, mengkonfigurasi dan menjalankan aplikasi yang ditulis. Server juga berisi layan-
¹¹ an lainnya, seperti *model extensible* data dan tampilan, mesin alur kerja atau mesin laporan.
¹² Namun, mereka adalah layanan OODO tidak secara khusus terkait dengan keamanan, dan
¹³ karena itu tidak dibahas secara rinci dalam dokumen ini.¹⁵

¹⁴ Server - ORM

¹⁵ Objek Relational Mapping ORM lapisan adalah salah satu fitur yang menonjol dari OODO
¹⁶ Server. Ini menyediakan fungsionalitas tambahan dan penting di atas server PostgreSQL.
¹⁷ Model Data dijelaskan dalam Python dan OODO menciptakan tabel database yang menda-
¹⁸ sari menggunakan ORM ini. Semua manfaat RDBMS seperti kendala yang unik, integritas
¹⁹ relasional atau query efisien digunakan dan dilengkapi dengan fleksibilitas Python. Misal-
²⁰ nya, kendala yang sewenang-wenang ditulis dengan Python dapat ditambahkan ke model
²¹ apapun. Mekanisme diperpanjang modular berbeda juga diberikan oleh OODO.¹⁶
²² Hal ini penting untuk memahami tanggung jawab ORM sebelum mencoba untuk by-pass
²³ dan untuk mengakses langsung database yang mendasari melalui query SQL baku. Bi-
²⁴ la menggunakan ORM, OODO dapat memastikan data tetap bebas dari korupsi apapun.
²⁵ Misalnya, sebuah modul dapat bereaksi terhadap penciptaan data dalam tabel tertentu.
²⁶ Perilaku ini dapat terjadi hanya jika pertanyaan melalui ORM. Layanan yang diberikan
²⁷ oleh ORM antara lain:¹⁷

- ²⁸ 1. konsistensi validasi oleh pemeriksaan validitas kuat.
- ²⁹ 2. menyediakan sebuah antarmuka pada objek (metode, referensi, ...) yang memungkinkan
³⁰ untuk merancang dan mengimplementasikan modul efisien.
- ³¹ 3. keamanan tingkat baris per pengguna dan kelompok; Rincian lebih lanjut tentang
³² pengguna dan kelompok pengguna diberikan dalam Pengguna bagian dan Peran Peng-
³³ guna.
- ³⁴ 4. tindakan yang kompleks pada sekelompok sumber daya.
- ³⁵ 5. Layanan warisan memungkinkan pemodelan baik sumber daya baru.

³⁶ Server - Web

³⁷ Lapisan web menawarkan antarmuka untuk berkomunikasi dengan browser standar. Dalam
³⁸ versi 6.1 dari OODO, web-client telah ditulis ulang dan diintegrasikan ke dalam server tier
³⁹ OODO. Lapisan web ini adalah aplikasi WSGI kompatibel berdasarkan Schieberegler. Ini
⁴⁰ menangani permintaan http biasa ke file server statis atau konten dinamis dan permintaan
⁴¹ JSON-RPC untuk RPC terbuat dari browser.¹⁸

¹⁵<https://doc.odoo.com/>

¹⁶<https://doc.odoo.com/>

¹⁷<https://doc.odoo.com/>

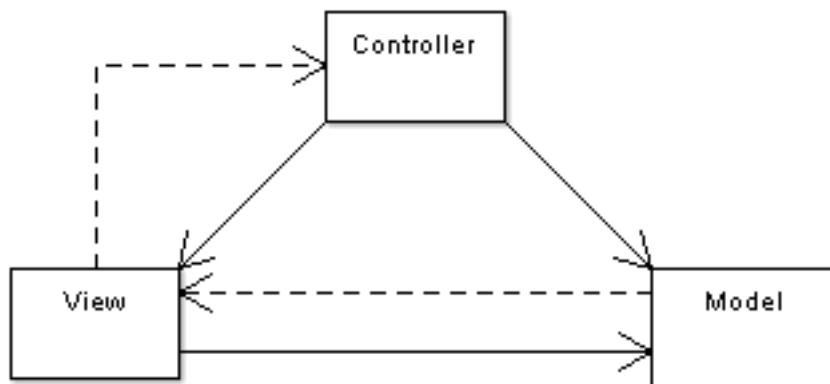
¹⁸<https://doc.odoo.com/>

1 Modul

2 Server OODOO merupakan *core* didalam arsitektur OODOO. Untuk setiap perusahaan, nilai
 3 OODOO terletak pada modul yang berbeda. Peran modul adalah untuk mengimplementasikan
 4 kebutuhan bisnis. Server adalah satu-satunya komponen yang diperlukan untuk menam-
 5 bahkan modul. Setiap rilis OODOO resmi mencakup banyak modul, dan ratusan modul yang
 6 tersedia. Contoh modul tersebut Akun, CRM, HR, Pemasaran, MRP, Sale, dll Klien.¹⁹ Mo-
 7 dul berperan sebagai *application logic* yang terkandung dalam *server-side* serta klien secara
 8 konseptual yang sederhana. Modul mengeluarkan permintaan ke server, mendapatkan data
 9 kembali dan menampilkan hasilnya (misalnya daftar pelanggan) dengan cara yang berbeda
 10 (seperti bentuk, daftar, kalender, ...). Setelah ada tindakan dari pengguna, modul akan
 11 mengirimkan permintaan untuk memodifikasi data ke server. Klien default OODOO ada-
 12 lah aplikasi JavaScript yang berjalan di browser yang berkomunikasi dengan server dengan
 13 menggunakan JSON-RPC.²⁰

14 Arsitektur MVC di OODOO

15 *Model-view-controller*(MVC) adalah pola arsitektur yang digunakan dalam rekomendasi per-
 16 angkat lunak". Dalam aplikasi komputer yang kompleks menyajikan banyak data kepada
 17 pengguna, kita sering ingin memisahkan data (*model*) dan *user interface*(*view*). Perubahan
 18 *user interface* tidak berdampak terhadap manajemen data, dan data dapat ditata kembali
 19 tanpa mengubah antarmuka pengguna. *Model-view-controller* memecahkan masalah tersebut
 20 dengan *decoupling* akses data dan logika bisnis dari penyajian data dan interaksi pengguna,
 21 dengan memperkenalkan komponen menengah yang disebut *controller*.²¹



Gambar 2.19: Diagram MVC

22 Pada tampilan diagram di atas, garis tebal untuk panah mulai dari *controller*, *view* dan
 23 *Model*, itu berarti bahwa *controller* memiliki akses lengkap terhadap *view* dan *Model*. Garis
 24 putus-putus untuk panah sebaliknya dari *view* ke *controller* berarti bahwa *view* memiliki
 25 akses terbatas terhadap *controller*.

26 Alasan dari desain ini adalah:²²

- 27 1. Dari View ke Model: *Model* mengirimkan pemberitahuan terhadap *view* ketika data
 28 yang telah dimodifikasi. *Model* ini tidak memerlukan *inner view* untuk melakukan
 29 operasi ini. Namun, *view* perlu mengakses bagian internal dari *Model*.
- 30 2. Dari View Controller: alasan mengapa *view* memiliki akses terbatas ke *controller*
 31 adalah karena ketergantungan dari *view* ke *controller* seminimal mungkin sehingga
 32 *controller* dapat diganti setiap saat.

¹⁹<https://doc.odoo.com/>

²⁰<https://doc.odoo.com/>

²¹<https://doc.odoo.com/>

²²<https://doc.odoo.com/>

- ¹ OODO mengikuti MVC semantik dengan
- ² 1. *Model*: Tabel PostgreSQL.
- ³ 2. *view*: *view* yang didefinisikan dalam file XML di OODO.
- ⁴ 3. *Controler*: Objek OODO.

1

BAB 3

2

ANALISIS

3 Pada bab ini, akan dijelaskan mengetai analisis kebutuhan dan fitur perangkat lunak, Dia-
4 gram pengembangan perangkat lunak, *use case* dari perangkat lunak serta diagram aktifitas
5 dari perangkat lunak.

6 **3.1 Analisis Kebutuhan dan Fitur Perangkat Lunak**

7 **3.1.1 Analisis Kebutuhan**

8 Pertama-tama modul BI ini terintegrasi dengan OODOO, sebelum menggunakan modul ini
9 maka pengguna diwajibkan untuk menginstall OODOO. Berikut ini *requirement component*
10 yang diperlukan untuk menginstall OODOO:

11 1. *installer* OODOO 8.0-20141128

12 2. PosgreSQL 9.3

13 3. Phyton 2.7

14 Sedangkan untuk minimum *requirement hardware*, yaitu:

15 1. Intel x86 atau x64

16 2. Minimum RAM 512 MB

17 3. Minimum ruang di hardisk 150 MB

18 4. Mendukung protokol TCP/IP

19 5. OS x86 atau x64 MAC, Linux, atau Windows

20 **3.1.2 Fitur Perangkat Lunak**

21 Perangkat Lunak ini akan memiliki fitur seperti gambar 3.1 :

22 1. BI *tool* menerima dan membaca *input source* berupa CSV/*text file*, dan modul OODOO
23 lain.

24 2. Dapat menentukan tipe dan format data

25 3. Dapat melakukan *Lookup*

26 4. Dapat melakukan agregat

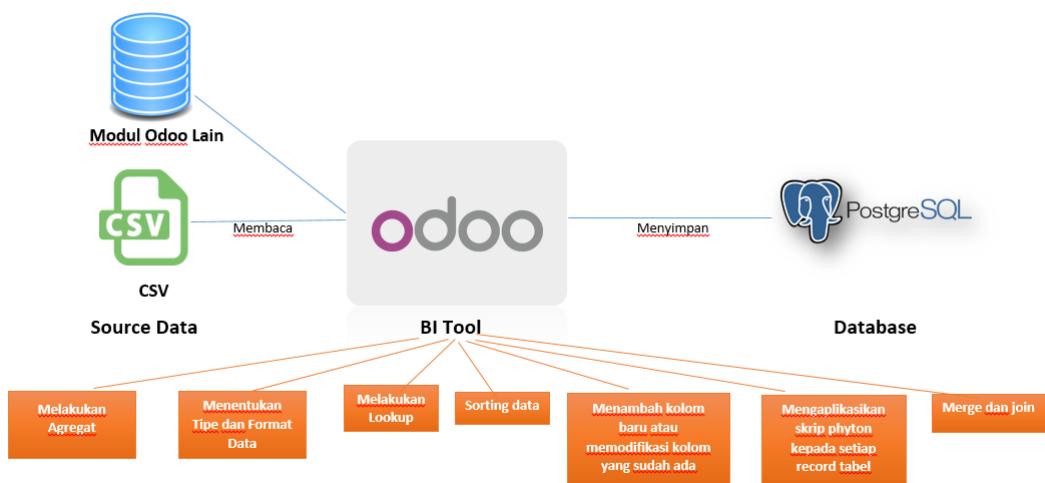
27 5. Dapat melakukan *sorting* data

28 6. Dapat menambah kolom baru atau memodifikasi kolom yang sudah ada

29 7. Dapat mengaplikasikan skrip phyton terhadap setiap baris *record*

30 8. Dapat melakukan *merge* dan *join*

31 9. Data yang telah diproses dapat disimpan dalam *database*



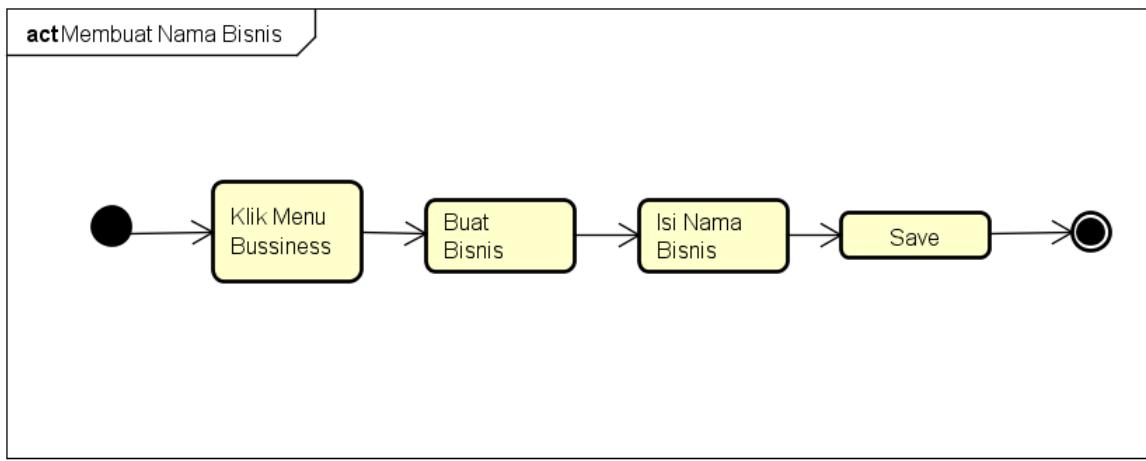
Gambar 3.1: Fitur BI tool

1 3.1.3 Diagram Aktifitas pada BI Tool

- 2 Pada subbab ini akan dibahas mengenai prosedur setiap aktifitas dari fitur yang diberikan oleh BI tool.

4 Membuat Bisnis Baru

- 5 Sebelum melangkah pada proses ETL pengguna diwajibkan membuat nama bisnis untuk membedakan setiap proses ETL. Berikut ini step-step untuk membuat bisnis baru.

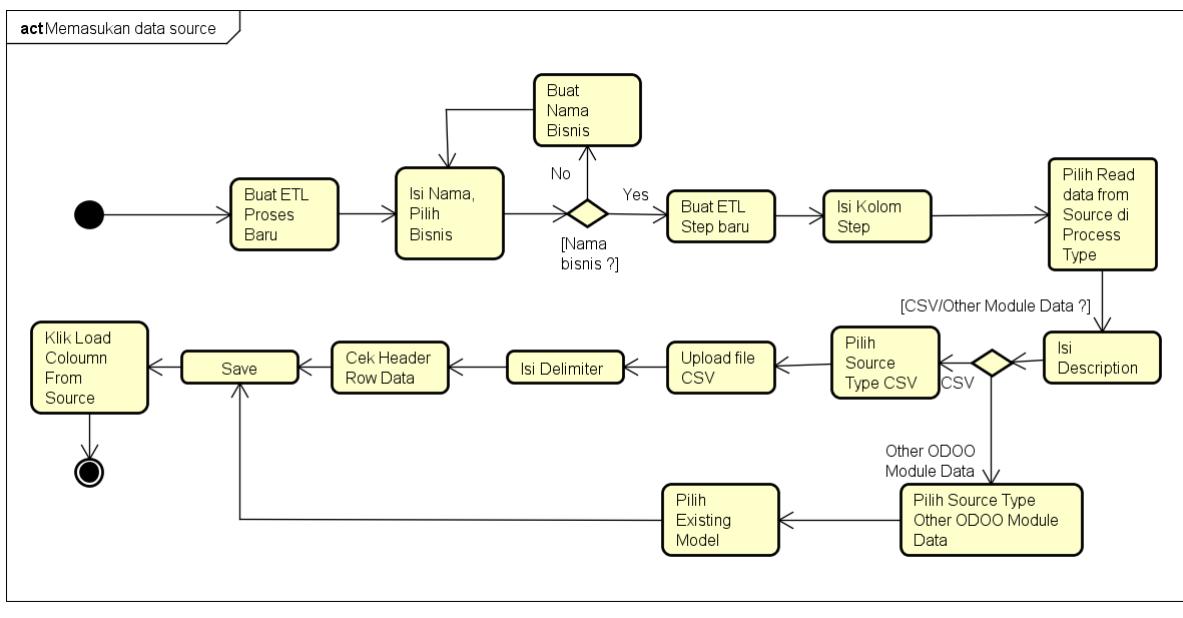


powered by Astah

Gambar 3.2: Prosedur Membuat Bisnis Baru

6

- 7 1. Pengguna mengklik menu *Businesses*.
- 8 2. Pengguna membuat nama bisnis baru.
- 9 3. Pengguna mengisi kolom nama bisnis.
- 10 4. Setelah selesai lalu pengguna mengklik tombol save.

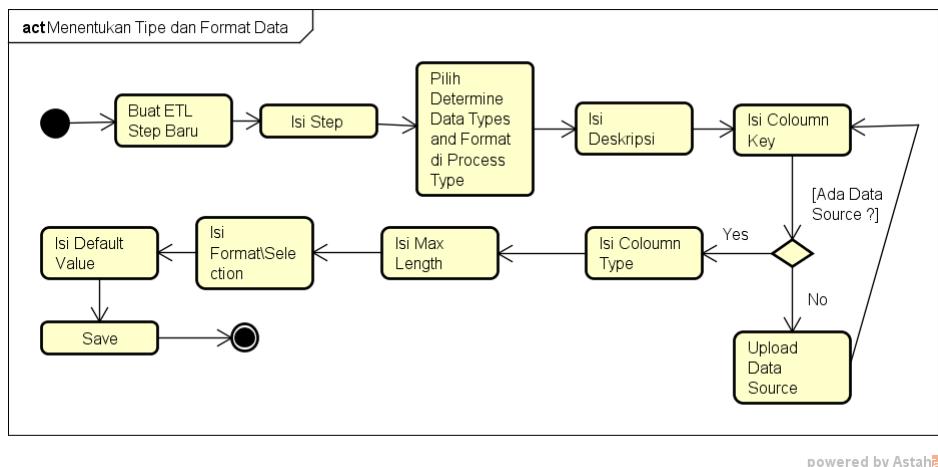
Gambar 3.3: Prosedur Memasukan *Data Source*

1 Memasukan *Data Source*

2 Pengguna dapat memasukan *data source* bertipe CSV/*teks file* atau modul OODOO lain dan
3 menggunakannya sebagai sumber data untuk tipe proses lain dari fitur BI. Berikut tahapan-
4 tahapan bagaimana memasukan data kedalam BI *tool*.

- 5 1. Pengguna mengakses menu ETL *Process* dan membuat proses ETL baru dengan me-
6 nuliskan nama pada kolom nama yang tersedia.
- 7 2. Selanjutnya pengguna memilih tipe bisnis yang telah dibuat sebelumnya, bila penggu-
8 na belum membuatnya maka pengguna dapat membuat nama bisnis terlebih dahulu
9 sebelum melangkah ke proses selanjutnya.
- 10 3. setelah kolom bisnis terisi maka pengguna membuat baru step etl
- 11 4. sesudahnya pengguna menuliskan nomer step untuk menentukan urutan *compile* oleh
12 BI *tool*
- 13 5. Pengguna memilih pilihan *read data from source* dikolom *process type* dilanjutkan
14 dengan menuliskan deskripsi prosesnya pada kolom *description*
- 15 6. Pengguna dapat memilih tipe *input CSV* atau Modul OODOO lain.
- 16 7. Jika pengguna memilih tipe data CSV maka pada kolom *source type* pilih CSV.
- 17 8. Selanjutnya, Pengguna meng-*upload* data CSV pada tempat yang disediakan
- 18 9. Pengguna lalu mengisi kolom *delimiter* dilanjutkan dengan menceklis kolom *header*
19 *row present?* bila baris pertama sumber data CSV adalah judul kolom.
- 20 10. Jika pengguna memilih modul OODOO lain maka pada kolom *source type* pilih *other*
21 *ODOO module data*.
- 22 11. Setelah itu, pengguna memilih modul yang telah terinstal dalam odoo yang akan
23 dijadikan *source*.
- 24 12. Setelah proses memilih *source* selesai, Pengguna mengklik tombol *save*, lalu mengklik
25 kembali tombol *load columns from source* untuk mengambil semua alamat kolom dalam
26 *source* untuk digunakan pada tipe proses selanjutnya.

1 Menentukan Tipe dan Format Data

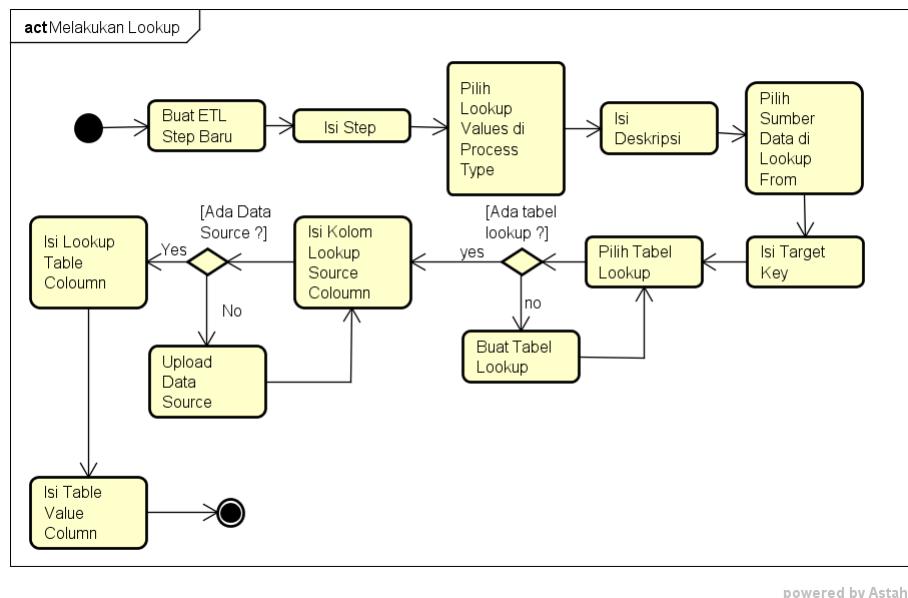


Gambar 3.4: Prosedur Menentukan Tipe dan Format Data

2 Pengguna dapat menentukan tipe dan format data sesuai keinginan, namun syaratnya
 3 adalah *data source* sudah di upload kedalam BI *tool*. Berikut ini prosedur menentukan tipe
 4 data serta memformat data.

- 5 1. Membuat step ETL baru pada menu *ETL process*.
- 6 2. kemudian menuliskan nomer step pada kolom step, karena step pertama di isi oleh
 7 *input data source* maka kolom step diisi oleh nomer yang lebih besar dari satu.
- 8 3. Memilih *determine data types and format* di kolom *Process type*, dilanjutkan dengan
 9 mengisi kolom deskripsi.
- 10 4. Selanjutnya mengisi *Column key*
- 11 5. Dilanjutkan dengan mengisi *column type*
- 12 6. lalu menuliskan *max length* untuk membatasi isi dari *field* tersebut.
- 13 7. Pengguna menuliskan *format/selection* dari agar format data seragam dan terstandarisasi.
- 14 8. Pengguna juga dapat mengisi kolom *default value* bila diperlukan.
- 15 9. Setelah proses selesai lalu pengguna mengklik tombol save dan semua settingan yang
 16 telah dimasukan pengguna akan tersimpan dalam *database*.

1 Melakukan *Lookup*

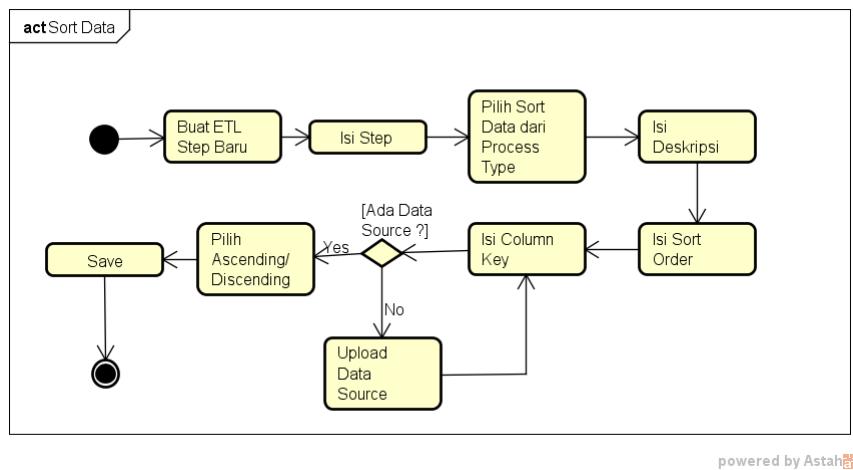


Gambar 3.5: Prosedur Melakukan *Lookup*

2 Pengguna dapat melakukan *lookup* dengan syarat *data source* telah dimasukan kedalam **3** *database BI tool*. Berikut pemaparan proses penggunaan *lookup*.

- 4** 1. Pengguna membuat ETL step baru dengan mengisi nomer urutan dikolom step.
- 5** 2. Pengguna memilih *process type lookup values* serta menulis deskripsi step tersebut.
- 6** 3. Pengguna dapat memilih melakukan proses *lookup* dengan tabel yang sudah ada didalam basis data BI tool atau menuliskan tabel baru pada kolom *lookup value* dan *result value*.
- 9** 4. pengguna menuliskan target *field* yang akan disi.
- 10** 5. kemudian memilih *value* kolom mana yang dijadikan acuan.
- 11** 6. setelah selesai pengguna mengklik tombol save untuk menyimpan settingan yang telah dimasukan sebelumnya.

1 Melakukan *Sorting Data*

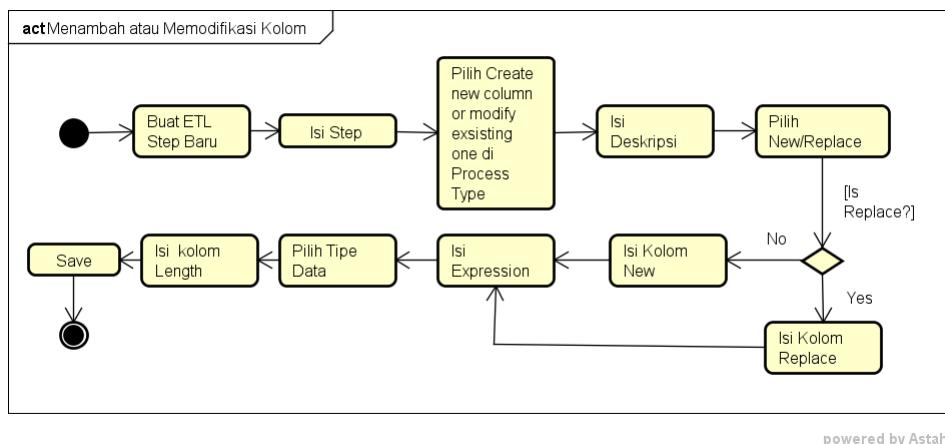


Gambar 3.6: Prosedur Melakukan *Sorting Data*

2 Pada bagian ini pengguna dapat melakukan *sorting data* dengan meng-input sumber data terlebih dahulu. Berikut penjelasan proses *sort order*

- 3 1. Pengguna membuat ETL step baru dengan mengisi nomer urutan dikolom step.
2. Selanjutnya memilih *process type sort data* dilanjutkan dengan mengisi deskripsi proses tersebut.
- 7 3. Pengguna menuliskan *sort order* sesuai dengan urutan *sort* yang terlebih dahulu di inginkan.
- 9 4. Pengguna menuliskan *column key* menandakan kolom *source* mana yang akan diurutkan.
- 11 5. Terakhir pengguna memilih *direction* dari *sort* yang di inginkan apakah *ascending* atau *descending*, lalu klik tombol *save* untuk menyimpan semua *setting* yang telah dilakukan.

14 Menambah atau Memodifikasi Kolom

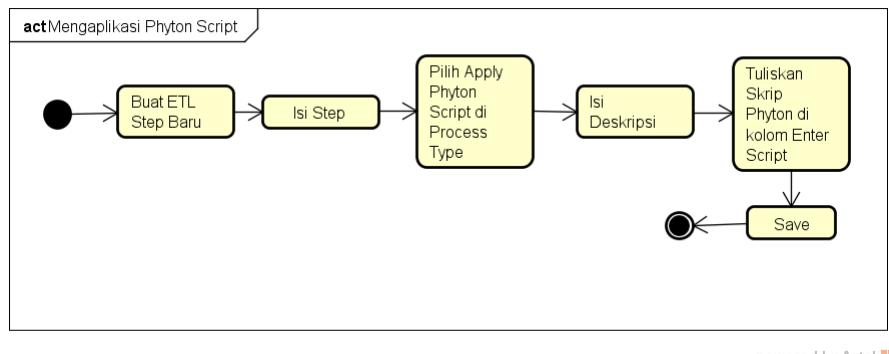


Gambar 3.7: Prosedur Menambah atau Memodifikasi Kolom

- 15 Pengguna dapat menambahkan kolom atau memodifikasi kolom yang tersedia dengan proses sebagai berikut.

1. Pengguna terlebih dahulu membuat ETL proses baru dan mengisi kolom step,
2. Selanjutnya memilih *create new column or modify existing one* pada kolom *process type* dan menuliskan deskripsi dari proses tersebut.
3. Pengguna menentukan apakan ingin menambahkan kolom baru atau memodifikasi kolom yang tersedia.
4. Jika memodifikasi kolom tentukan kolom yang akan dimodifikasi.
5. Jika membuat kolom baru pengguna menuliskan nama kolom baru.
6. Setelah itu pengguna dapat menuliskan fungsi python untuk perhitungan atau tujuan lain pada kolom *expression*.
7. Pengguna menentukan tipe data kolom tersebut.
8. Pengguna dapat menentukan panjang data yang akan dimasukan pada kolom *length*.

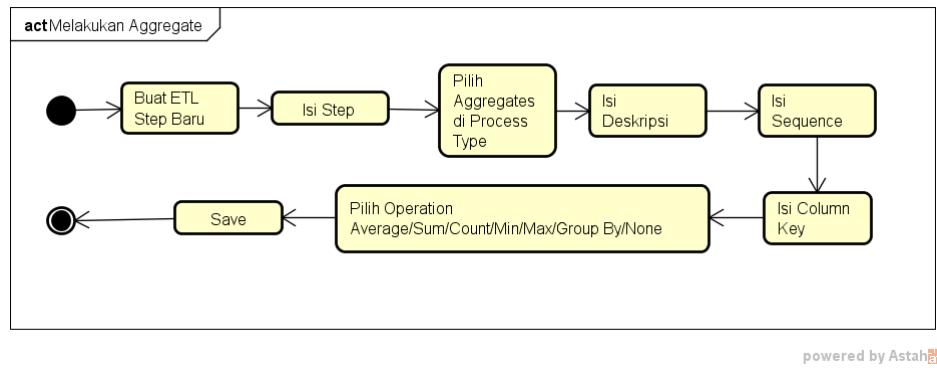
12 Mengaplikasikan Phyton Script



Gambar 3.8: Prosedur Mengaplikasikan Phyton Script

13. Pengguna dapat menjalankan skrip phyton yang berlaku untuk setiap baris data yang pada *source*. Berikut proses memasukan skrip phyton pada BI *tool*.
14. 1. Pengguna membuat ETL step baru dengan mengisi nomer urutan dikolom step.
15. 2. Selanjutnya memilih *process type apply phyton script to each data line* dilanjutkan dengan mengisi deskripsi proses tersebut.
16. 3. Setelah itu pengguna dapat langsung memasukan skrip phyton pada kolom *enter script* pada perangkat lunak.
17. 4. Setelah selesai pengguna mengklik tombol save untuk menyimpan semua *setting*.

1 Melakukan Aggregat

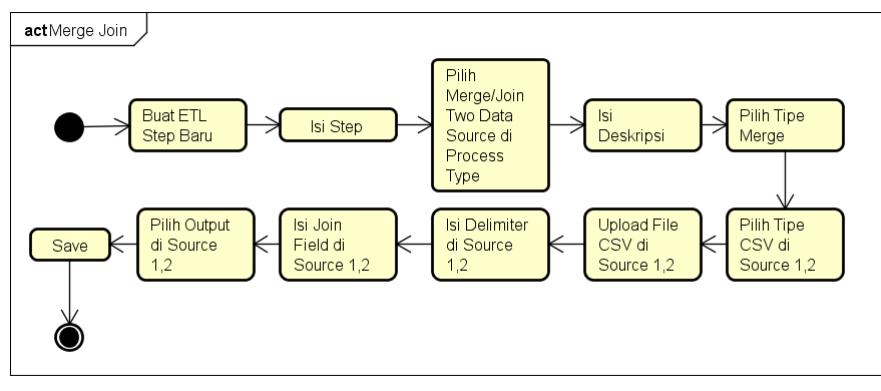


Gambar 3.9: Prosedur Melakukan *Aggregate*

2 Pengguna dapat melakukan agregat terhadap *data source*. Berikut pemaparan proses agregat.

- 4 1. Pengguna membuat ETL step baru dengan mengisi nomer urutan dikolom step.
- 5 2. Pengguna memilih *aggregates* pada kolom *process type* dan menuliskan deskripsi mengenai proses tersebut.
- 7 3. Pengguna menuliskan urutan dari pengajaran agregat pada kolom *sequence*.
- 8 4. Pengguna memilih kolom yang akan di agregat.
- 9 5. Pengguna memilih operasi yang dilakukan apakah *count*, *max*, *min*, *sum*, *average*, *group by* ataupun *none*.
- 11 6. Setelah selesai pengguna mengklik tombol *save*.

12 Melakukan *Merge Join*



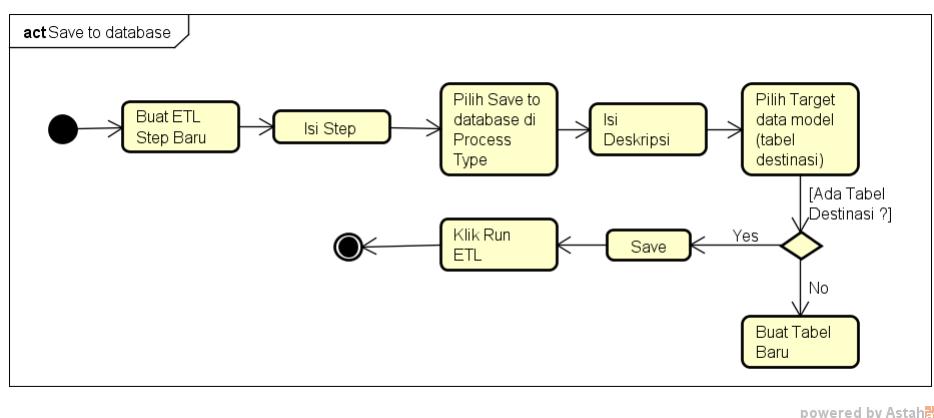
Gambar 3.10: Prosedur Melakukan *Merge Join* dari dua *data source*

13 Pengguna dapat melakukan *merging* terhadap dua *data source* sekaligus. Berikut pemaparan prosesnya.

- 15 1. Pengguna membuat ETL step baru dengan mengisi nomer urutan dikolom step.
- 16 2. Setelah itu memilih *merge/join two data source* pada kolom *process type* dan menuangkan deskripsi proses tersebut pada kolom *description*.

- 1 3. Pengguna memilih tipe *merge* yang akan digunakan, yaitu *left outer join*, *inner join*, atau *right outer join*.
- 2 4. Pengguna menentukan tipe *source*.
- 3 5. Pengguna *upload data source* jika tipe masukannya CSV.
- 4 6. Pengguna mengisi kolom *delimiter* jika tipe masukannya CSV.
- 5 7. Setelah *upload* selesai maka kolom pada *source* akan terbaca otomatis oleh BI *tool*.
- 6 8. Setelah *source* pertama selesai, pengguna melakukan step yang sama terhadap *source* ke=2.
- 7 9. Setelah kedua *source* sudah terbaca oleh BI *tool*, selanjutnya pengguna menuliskan *join field* pada kedua kolom *source* tersebut.
- 8 10. Pengguna dapat menentukan *output* kolom dengan menceklis mana saja yang akan menjadi *output* dalam tabel destinasi pada bagian *output* dalam BI *tool*.
- 9 11. Setelah semua selesai maka pengguna mengklik tombol *save*.

14 ***Save to Database***



Gambar 3.11: Prosedur Menyimpan Data dalam *Database*

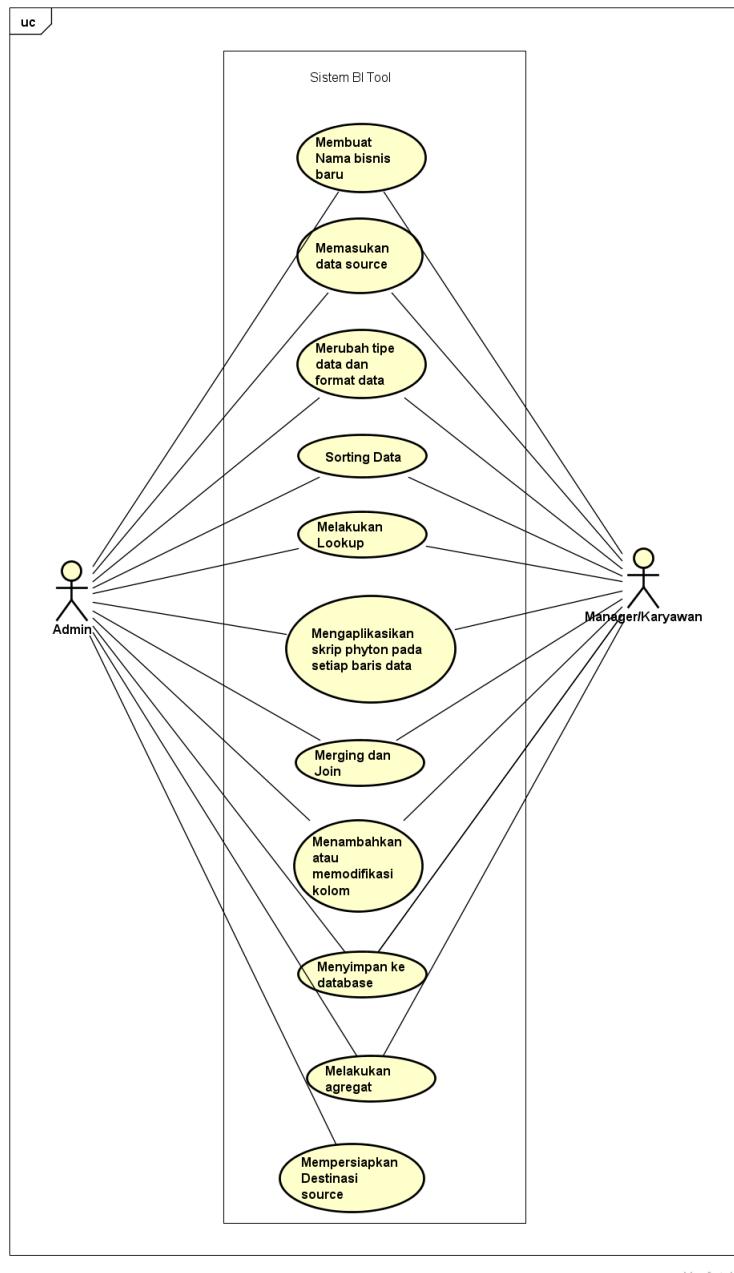
- 15 Setelah semua tipe proses dilalui maka pengguna perlu menyimpan data hasil proses kedalam *database* BI *tool*. Namun, pastikan semua *process type* telah terisi dengan benar, bila tidak hasil proses ETL tidak akan tersimpan dalam *database*. Untuk menyimpan data hasil proses kedalam tabel destinasi dalam *database* syaratnya perlu minimal satu tipe proses *read data from source* atau *merge join* yang meminta *input data source* pada pengguna. Berikut pemaparan lebih lanjut mengenai prosesnya.

- 21 1. Pengguna membuat ETL step baru dengan mengisi nomer urutan dikolom step.
- 22 2. Selanjutnya pengguna memilih *save to database* pada kolom *process type* dan menuliskan deskripsi singkat mengenai proses tersebut.
- 23 3. Pengguna menentukan tabel destinasi tempat penyimpanan, dalam istilah OODOO disebut *target data model*.
- 24 4. Kolom destinasi akan terbaca secara otomatis oleh BI *tool* setelah menentukan tabel destinasi/*target data model*.

5. Pengguna melakukan *mapping* kolom dari *source* mana yang akan masuk dalam tabel destinasi.
6. Setelah semua selesai klik *save*.
7. Lalu, pengguna mengklik tombol *Run ETL* untuk menjalankan semua tipe proses ETL dan menyimpannya dalam *database*

3.2 Permodelan Tool

- Berikut diagram use case berserta skenario yang tertera pada gambar 3.10



Gambar 3.12: Diagram use case BI tool

1. Skenario Membuat Bisnis Baru

9

- 10 Deskripsi : Kegiatan membuat nama bisnis.

1 Aktor : Admin, manajer
2 Prakondisi : -
3 Skenario :
4 – Admin, manajer dapat membuat nama bisnis baru
5 2. Skenario Memasukan *Data Source*
6 Deskripsi : Kegiatan meng-*upload data source* berupa *file CSV* atau teks.
7 Aktor : Admin, Manajer
8 Prakondisi : -
9 Skenario :
10 – Admin, manajer meng-*upload CSV* atau file teks melalui menu ETL *process*.
11 3. Skenario Merubah Tipe dan Format Data
12 Deskripsi : Kegiatan merubah tipe data dan format data.
13 Aktor : Admin, Manajer
14 Prakondisi : *data source* telah *diinput* sebelumnya.
15 Skenario :
16 – Admin, manajer dapat menentukan tipe data, format data, *default value*
17 maupun panjang data agar seragam.
18 4. Skenario *Sorting Data*
19 Deskripsi : Kegiatan mengurutkan data terhadap *data source*.
20 Aktor : Admin, Manajer
21 Prakondisi : *data source* telah *diinput* sebelumnya
22 Skenario :
23 – Admin, manajer dapat mengurutkan data sesuai *order penulisannya* dan da-
24 pat menentukan arah pengurutan secara menaik atau menurun.
25 5. Skenario Melakukan *Lookup*
26 Deskripsi : Kegiatan melakukan *lookup* terhadap data *input source*.
27 Aktor : Admin, Manajer
28 Prakondisi : *data source* dan data tabel yang akan *dilookup* telah *diinput* sebe-
29 lunya jika ingin memilih *lookup* dari *existing table*
30 Skenario :
31 – Admin, manajer dapat melakukan *lookup* terhadap *data source* dengan data
32 tabel yang telah ada dalam *database* maupun data tabel yang dimasukan
33 bersamaan pada proses setting *input program lookup*.
34 – Admin, manajer dapat menuliskan lebih dari satu *column key* pada tabel
35 *source* dan tabel *lookup*
36 6. Skenario Melakukan *Merging Join*
37 Deskripsi : Kegiatan melakukan *Merging/join* terhadap dua *data sources*.
38 Aktor : Admin, Manajer
39 Prakondisi : -
40 Skenario :

- 1 – Admin, manajer dapat melakukan *Merging/join* dengan memasukan dua
- 2 sumber data yang mempunyai *key/join field* yang sama.
- 3 – Admin, manajer dapat menentukan tipe *merge* yaitu *left outer join*, *inner*
- 4 *join*, maupun *right outer join*.
- 5 – Admin, manajer dapat menentukan kolom mana yang akan menjadi *output*
- 6 dari proses ini.

7 7. Skenario Menambahkan atau Memodifikasi Kolom

8 Deskripsi : Kegiatan menambahkan kolom baru atau memodifikasi kolom yang
9 sudah ada pada *data sources*.

10 Aktor : Admin, Manajer

11 Prakondisi : -

12 Skenario : *data source* telah *diinput* sebelumnya

- 13 – Admin, manajer dapat menambahkan kolom maupun memodifikasi kolom
- 14 yang sudah ada sebelumnya.
- 15 – Admin, manajer dapat menentukan nama kolom baru jika memilih untuk
- 16 menambah kolom baru.
- 17 – Admin, manajer dapat memilih kolom yang akan di *replace* dengan pada
- 18 kolom *replace* pada BI *tool*.
- 19 – Admin, manajer dapat menentukan tipe data , panjang data serta serta me-
- 20 nuliskan *expression* dengan *function* yang dimiliki phyton untuk melakukan
- 21 perhitungan yang hasilnya akan dimasukan kedalam kolom tersebut.

22 8. Skenario Melakukan agregat

23 Deskripsi : Kegiatan melakukan agregat.

24 Aktor : Admin, Manajer

25 Prakondisi : -

26 Skenario : *data source* telah *diinput* sebelumnya

- 27 – Admin, manajer dapat melakukan agregat dengan menentukan *column key*
- 28 yang akan di aggregat, menentukan urutan kolom yang akan di aggregat, dan
- 29 menentukan operasi aggregat yang dilakukan apakah *sum*, *min*, *max*, *average*,
- 30 *count*, *group by* atau *none*.

31 9. Skenario Menyimpan ke *database*

32 Deskripsi : Kegiatan menyimpan hasil proses ETL kedalam *database*.

33 Aktor : Admin, Manajer

34 Prakondisi : *data source* telah *diinput* sebelumnya

35 Skenario :

- 36 – Admin, manajer dapat melakukan menyimpan hasil proses ETL dan mela-
- 37 kukan *mapping* kolom di data sumber mana yang akan masuk kedalam tabel
- 38 destinasi.

39 10. Skenario Mempersiapkan *Destination source*

40 Deskripsi : Kegiatan mempersiapkan *destination source* untuk menyimpan hasil
41 proses ETL.

42 Aktor : Admin

43 Prakondisi : -

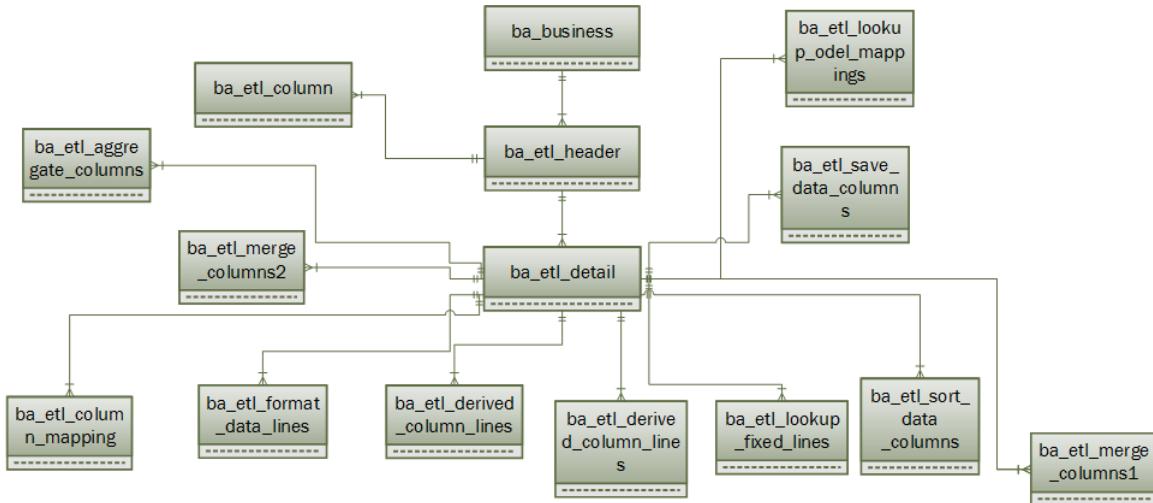
44 Skenario :

- 45 – Admin mempunyai kuasa lebih dalam mempersiapkan tabel destinasi yang
- 46 nanti akan dipakai sebagai penyimpanan hasil proses ETL.

3.3 Pemodelan Basisdata

3.3.1 Perancangan Basisdata

Berikut ini merupakan model perancangan basisdata dari BI *tool*.



Gambar 3.13: Struktur tabel BI *tool*

Berikut ini penjelasan mengenai pemodelan basisdata dari BI *tool*

1. **ba_business** : Tujuan pembuatan tabel ini adalah supaya banyak perusahaan/bisnis yang dapat menggunakan *tool* ini (*multicompany/multiclient*).
2. **ba_etl_header** : Berfungsi sebagai list proses ETL yang ada di bawah perusahaan tersebut.
3. **ba_etl_detail** : Menyimpan detail setiap proses ETL di bawah perusahaan tersebut.
4. **ba_etl_columns**: Kolom-kolom yang terlibat di dalam proses ETL, baik dari file sumber data maupun yang ditambahkan di tengah proses.
5. **ba_etl_columns_mapping** : Detil *mapping* kolom dari data *input* ke *output*. Dimana user bisa memilih untuk tetap memakai nama kolom lama atau mengubahnya.
6. **ba_etl_format_data_lines** : Khusus tipe ETL *format data* menyimpan detail informasi, tipe, dll dari setiap kolom.
7. **ba_etl_sort_data_columns** : Khusus tipe ETL *sort data* menyimpan detail informasi kolom dan arah sorting.
8. **ba_etl_derived_column_lines** : Khusus tipe ETL *derived column* menyimpan detail formula untuk menghasilkan *derived column*.
9. **ba_etl_lookup_fixed_lines** : Khusus tipe ETL *lookup* menyimpan bila *lookup* diambil dari himpunan pilihan *fixed*, tabel ini menampung piluhan-pilihan tersebut.
10. **ba_etl_save_data_columns** : Khusus tipe ETL *save data* menyimpan *mapping* kolom antara hasil dari proses ETL dengan *field* di tabel.
11. **ba_etl_lookup_podel_mappings** : Khusus tipe ETL *lookup* menyimpan bila *lookup* diambil dari tabel lain, tabel ini menampung pasangan *field* di data proses dengan yang di tabel *lookup*.

- 1 12. **ba_etl_merge_columns1** : Khusus tipe ETL *merge* menyimpan informasi *merge*
2 dari sumber pertama.
- 3 13. **ba_etl_merge_columns2** : Khusus tipe ETL *merge* menyimpan informasi merge
4 dari sumber kedua
- 5 14. **ba_etl_aggregate_columns** : Khusus tipe ETL *aggregate* menyimpan detil in-
6 formasi proses *aggregate*

¹

BAB 4

²

PERANCANGAN

³ Berdasarkan analisis pada Bab 3, pada bab ini akan dipaparkan mengenai perancangan
⁴ lojik basisdata, perancangan fisik basisdata, perancangan kelas MVC pada ODOO, diagram
⁵ kelas, perancangan antar muka BI *tool*, rancangan *method-method* utama.

⁶ 4.1 Perancangan Lojik Basisdata

⁷ Setelah pada bab 3 dijelaskan mengenai pemodelan basis data BI *tool*, pada bagian ini akan
⁸ digambarkan secara jelas rancangan struktur basisdata dari BI *tool*. Rancangan basisdata
⁹ ini merupakan implementasi dari skema *star* menempatkan ba_etl_detail sebagai *fact table*.
¹⁰ Keuntungan menerapkan skema *star* adalah :

¹¹ 1. Lebih mudah dipahami sehingga memudahkan pengguna jika ingin mengembangkan
¹² BI *tool* ini lebih lanjut.

¹³ 2. Secara teknis memudahkan *developer* jika ingin mengembangkan fitur baru pada BI
¹⁴ *tool* ini.

¹⁵ 3. Efisien dalam mengakses data sehingga memudahkan untuk menampilkannya.

Gambar 4.1: Diagram Relational pada Basisdata BI *tool*

4.2 Perancangan Fisik Basisdata

- 2 Pada bagian sebelumnya telah digambarkan rancangan lojik basisdata BI *tool*, berikut ini
- 3 penjelasan detil kolom-kolom dalam tabel tersebut.

Tabel 4.1: Tabel *ba_bussiness*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
name	VARCHAR	200	NOT NULL

Tabel 4.2: Tabel *ba_etl_header*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
bussiness_id	MANY2ONE ba_business		NOT NULL; FOREIGN KEY
name	VARCHAR	200	NOT NULL
is_active	BOOLEAN		DEFAULT TRUE

Tabel 4.3: Tabel *ba_etl_detail*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
header_id	MANY2ONE ba_etl_header		NOT NULL; FOREIGN KEY
process_type	ENUM(read_data, format_data, lo- okup, sort_data, derived_column, script, merge, aggregate, sort_data, save_ data)		NOT NULL
name	VARCHAR		NOT NULL
sequence	INTEGER		NOT NULL
read_data_source_type	ENUM(CSV)	20	
read_data_csv_trial_file	BINARY		
read_data_csv_delimiter	VARCHAR	2	
read_data_csv_auto_read	BOOLEAN		
read_data_csv_auto_read_path	VARCHAR		
read_data_csv_is_header_present	BOOLEAN		DEFAULT TRUE
script_text	TEXT		
save_data_model_id	MANY2ONE ir_model		
lookup_source_key_column	MANY2ONE ba_etl_columns		
lookup_target_key_column	varchar	64	
lookup_from	ENUM(fixed, existing_table)		
lookup_model_id	MANY2ONE ir_model		
lookup_model_key_column	VARCHAR	64	
lookup_model_value_column	VARCHAR	64	
merge_source_type_1	ENUM(CSV)		
merge_source_type_2	ENUM(CSV)		
merge_csv_trial_file_1	BINARY		
merge_csv_trial_file_2	BINARY		
merge_csv_delimiter_1	VARCHAR	3	
merge_csv_delimiter_2	VARCHAR	3	
merge_type	ENUM(left, right, inner)		

Tabel 4.4: Tabel *ba_etl_columns*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
header_id	MANY2ONE ba_etl_header		FOREIGN KEY
name	VARCHAR	64	

Tabel 4.5: Tabel *ba_etl_columns_mapping*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
header_id	MANY2ONE ba_etl_detail		NOT NULL; FOREIGN KEY
source_column	VARCHAR	64	
destination_column	VARCHAR	64	

Tabel 4.6: Tabel *ba_etl_format_data_lines*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
header_id	MANY2ONE ba_etl_detail		NOT NULL; FOREIGN KEY
column_key	MANY2ONE ba_etl_columns		NOT NULL
column_type	ENUM(char, integer, float, date, datetime, boolean, selection)		
string_length	INTEGER		
column_format	VARCHAR	50	
column_default	VARCHAR	500	

Tabel 4.7: Tabel *ba_etl_sort_data_columns*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
header_id	MANY2ONE ba_etl_detail		NOT NULL; FOREIGN KEY
sequence	INTEGER		NOT NULL
column_key	MANY2ONE ba_etl_columns		
sort_order	ENUM(asc, desc)		DEFAULT ASC

Tabel 4.8: Tabel *ba_etl_derived_column_lines*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
header_id	MANY2ONE ba_etl_detail		NOT NULL; FOREIGN KEY
column_key	MANY2ONE ba_etl_columns		
new_or_modify	ENUM(new, modify)		DEFAULT NEW
new_column_key	VARCHAR	64	
expression	TEXT		
data_type	ENUM(char, integer, float, date, datetime, boolean, selection)		
length	INTEGER		

Tabel 4.9: Tabel *ba_etl_lookup_fixed_lines*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
header_id	MANY2ONE ba_etl_detail		NOT NULL; FOREIGN KEY
lookup	VARCHAR	500	
value	VARCHAR	500	

Tabel 4.10: Tabel *ba_etl_save_data_column*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
header_id	MANY2ONE ba_etl_detail		
source_column	MANY2ONE ba_etl_column		
destination_column	VARCHAR	64	

Tabel 4.11: Tabel *ba_etl_lookup_model_mappings*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
header_id	MANY2ONE ba_etl_detail		
source_column	MANY2ONE ba_etl_column		
destination_column	VARCHAR	64	

Tabel 4.12: Tabel *ba_etl_merge_columns_1*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
header_id	MANY2ONE ba_etl_detail		
source_column	VARCHAR	64	
destination_column	VARCHAR	64	
is_included	BOOLEAN		DEFAULT 1
join_field	VARCHAR	64	

Tabel 4.13: Tabel *ba_etl_merge_columns_2*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
header_id	MANY2ONE ba_etl_detail		
source_column	VARCHAR	64	
destination_column	VARCHAR	64	
is_included	BOOLEAN		DEFAULT 1
join_field	VARCHAR	64	

Tabel 4.14: Tabel *ba_etl_aggregate_columns*

Field	Tipe	Panjang	Constraint
id	NUMBER	10	NOT NULL; PRIMARY KEY
header_id	MANY2ONE ba_etl_detail		
sequence	INTEGER		
column_key	MANY2ONE ba_etl_columns		
operation	ENUM(None, count, sum, avg, max, min, group_by)		

4.3 Diagram Kelas

- 2 Berikut ini merupakan gambaran diagram kelas dari BI *tool*. Setiap kelas pada diagram ini meng-*extend* kelas osv yang merupakan bawaan dari *framework* OODOO.

Tabel 4.15: Tabel Kelas *ba_bussiness*

Atribut	
Nama atribut	Tipe Data
name	CHAR(200)

Tabel 4.16: Tabel Kelas *ba_etl_header*

Atribut	
Nama atribut	Tipe Data
name	CHAR(100)
business_id	MANY2ONE('ba.business')
is_active	BOOLEAN
etl_columns	ONE2MANY('ba.etl.columns')
etl_detail	ONE2MANY('ba.etl.detail')
Method	
action_run_etl()	
action_open_etl_run()	
action_fillin_columns()	

Tabel 4.17: Tabel Kelas *ba_etl_columns*

Atribut	
Nama atribut	Tipe Data
header_id	MANY2ONE('ba.etl.header')
name	CHAR

Tabel 4.18: Tabel Kelas *ba_etl_detail*

Atribut	
Nama atribut	Tipe Data
header_id	MANY2ONE('ba.etl.header')
process_type	selection
name	CHAR(255)
sequence	INT
column_mapping	ONE2MANY('ba.el.column.mapping')
read_data_source_type	SELECTION
read_data_csv_trial_file	BINARY
read_data_csv_delimiter	CHAR
read_data_csv_auto_read	BOOLEAN
read_data_csv_auto_read_path	CHAR
read_data_csv_is_header_present	BOOLEAN
read_data_model_id	MANY2ONE('ir.model')
format_data_type_lines	ONE2MANY('ba.etl.format.data.lines')
script_text	TEXT
save_data_model_id	MANY2ONE('ir.model')
save_data_columns	ONE2MANY('ba.etl.save.data.columns')
sort_data_columns	ONE2MANY('ba.etl.sort.data.column')
derived_column_lines	ONE2MANY('ba.etl.derived.column')
lookup_source_key_column	MANY2ONE('ba.etl.columns')
lookup_target_key_column	CHAR
lookup_from	SELECTION
lookup_model_id	MANY2ONE('ir.model')
lookup_model_value_column	MANY2ONE(ir.model.fields)
lookup_fixed_values	ONE2MANY('ba.etl.lookup.fixed.lines')
lookup_model_mappings	ONE2MANY('ba.etl.lookup.model.mappings')
merge_source_type_1	SELECTION
merge_source_type_2	SELECTION
merge_csv_trial_file_1	BINARY
merge_csv_trial_file_2	BINARY
merge_csv_delimiter_1	CHAR
merge_csv_delimiter_2	CHAR
merge_type	SELECTION
merge_column_1	ONE2MANY('ba.etl.merge.columns1')
merge_column_2	ONE2MANY('ba.etl.merge.columns2')
Method	
get_columns_from_csv(csv_content, delimiter)	
onchange_header_id(header_id)	
onchange_csv_trial_file(read_data_csv_trial_file, delimiter)	
onchange_csv_trial_file1(merge_csv_trial_file_1, merge_csv_delimiter_1)	
onchange_csv_trial_file2(merge_csv_trial_file_2, merge_csv_delimiter_2)	
onchange_read_data_model(read_data_model_id)	
get_etl_columns(header_id)	
onchange_merge_source_type_1(header_id, merge_source_type_1)	
onchange_save_data_model(save_data_model_id)	
onchange_lookup_model_id(lookup_model_id)	

Tabel 4.19: Tabel Kelas *ba_etl_columns_mapping*

Atribut	
Nama atribut	Tipe Data
header_id	MANY2ONE('ba.etl.detail')
source_column	CHAR(255)
destination_column	CHAR
Method	
constraint_datakey_name()	

Tabel 4.20: Tabel Kelas *ba_etl_format_data_lines*

Atribut	
Nama atribut	Tipe Data
header_id	MANY2ONE('ba.etl.detail')
column_key	MANY2ONE('ba.etl.column')
column_type	SELECTION
string_length	INT
column_format	CHAR(50)
column_default	CHAR(500)

Tabel 4.21: Tabel Kelas *ba_etl_sort_data_columns*

Atribut	
Nama atribut	Tipe Data
header_id	MANY2ONE('ba.etl.detail')
sequence	INT
column_key	MANY2ONE('ba.etl.column')
sort_order	SELECTION

Tabel 4.22: Tabel Kelas *ba_etl_derived_column_lines*

Atribut	
Nama atribut	Tipe Data
header_id	MANY2ONE('ba.etl.detail')
column_key	MANY2ONE('ba.etl.column')
new_or_modify	SELECTION
new_column_key	CHAR(64)
expression	TEXT
data_type	SELECTION
length	INT

Tabel 4.23: Tabel Kelas *ba_etl_save_data_columns*

Atribut	
Nama atribut	Tipe Data
header_id	MANY2ONE('ba.etl.detail')
source_column	MANY2ONE('ba.etl.column')
destination	CHAR(64)

Tabel 4.24: Tabel Kelas *ba_etl_lookup_fixed_lines*

Atribut	
Nama atribut	Tipe Data
header_id	MANY2ONE('ba.etl.detail')
lookup	CHAR
value	CHAR

Tabel 4.25: Tabel Kelas *ba_etl_lookup_model_mappings*

Atribut	
Nama atribut	Tipe Data
header_id	MANY2ONE('ba.etl.detail')
source_column	MANY2ONE('ba.etl.column')
model_column	MANY2ONE('ir.model.fields')

Tabel 4.26: Tabel Kelas *ba_etl_merge_columns1*

Atribut	
Nama atribut	Tipe Data
header_id	MANY2ONE('ba.etl.detail')
source_column	CHAR(64)
destination_column	CHAR(64)
is_included	BOOLEAN
join_field	CHAR

Tabel 4.27: Tabel Kelas *ba_etl_merge_columns2*

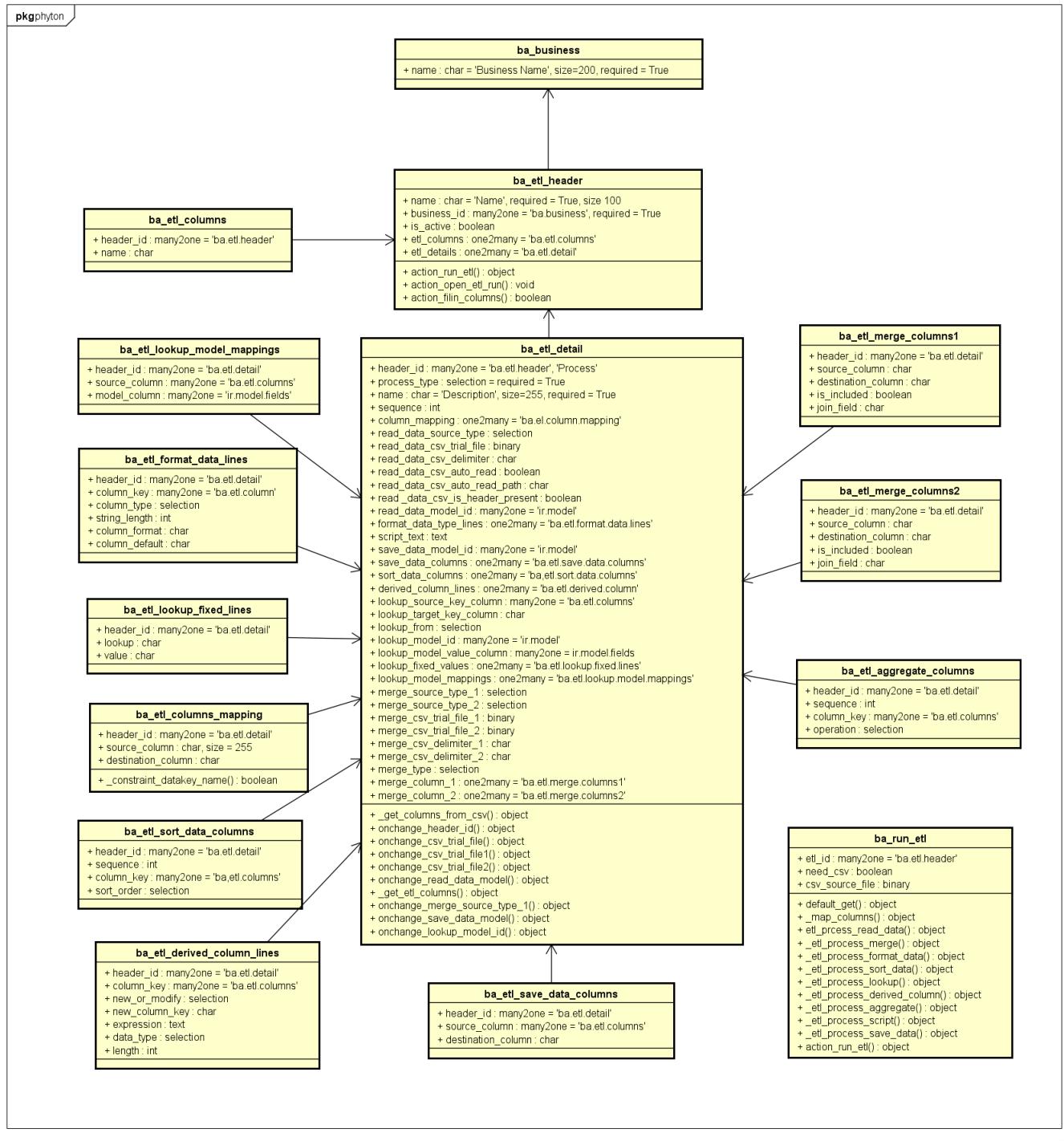
Atribut	
Nama atribut	Tipe Data
header_id	MANY2ONE('ba.etl.detail')
source_column	CHAR(64)
destination_column	CHAR(64)
is_included	BOOLEAN
join_field	CHAR

Tabel 4.28: Tabel Kelas *ba_etl_aggregate_columns*

Atribut	
Nama atribut	Tipe Data
header_id	MANY2ONE('ba.etl.detail')
sequence	INT
column_key	MANY2ONE('ba.etl.columns')
operation	SELECTION

Tabel 4.29: Tabel Kelas *ba_run_etl*

Atribut	
Nama atribut	Tipe Data
etl_id	MANY2ONE('ba.etl.header')
need_csv	BOOLEAN
csv_source_file	BINARY
Method	
default_get(fields)	
map_columns(source_data, columns_mapping)	
etl_process_read_data(step_info, raw_data)	
etl_process_merge(step_info, raw_data)	
etl_process_format_data(step_info, raw_data)	
etl_process_sort_data(step_info, raw_data)	
etl_process_lookup(step_info, raw_data)	
etl_process_derived_column(step_info, raw_data)	
etl_process_aggregate(step_info, raw_data)	
etl_process_script(step_info, raw_data)	
etl_process_save_data(step_info, raw_data)	
action_run_etl(etl_id)	



Gambar 4.2: Struktur kelas diagram BI tool

¹ **BAB 5**

² **IMPLEMENTASI DAN PENGUJIAN**

³ Pada bagian ini akan dijelaskan mengenai lingkungan implementasi perangkat keras maupun perangkat lunak. Serta implementasi basisdata BI *tool* beserta tampilan antar muka.
⁴ Terakhir akan dibahas mengenai pengujian pada perangkat lunak ini.
⁵

⁶ **5.1 Lingkungan Implementasi**

⁷ **5.1.1 Lingkungan Perangkat Keras**

⁸ Dalam mengembangkan perangkat ini, digunakan spesifikasi perangkat keras sebagai berikut:
⁹

- ¹⁰ • Processor : Intel Core i7 2.4 Ghz
- ¹¹ • *Memory* : 8 GB
- ¹² • *Hardisk* : 640 GB
- ¹³ • VGA : Nvidia GeForce 540M
- ¹⁴ • *keyboard* dan *mouse standard*

¹⁵ **5.1.2 Lingkungan Perangkat Lunak**

¹⁶ Untuk pengembangan perangkat lunak BI *tool*, digunakan spesifikasi sebagai berikut:

- ¹⁷ • *Tools*: OODOO 8.0-20141128
- ¹⁸ • Bahasa Pemograman Phyton 2.7
- ¹⁹ • *Database management system* PosgreSQL 9.3
- ²⁰ • *Operating system*: windows 8

²¹ **5.2 Implementasi Tabel Basisdata**

²² **5.2.1 Tabel Basisdata ETL dan Bisnis**

²³ Berikut implementasi tabel basisdata yang terlibat langsung dalam proses ETL beserta pendukungannya.
²⁴

- ²⁵ • Tabel *ba_business*
²⁶ Tabel ini digunakan untuk mencatat semua bisnis yang menggunakan perangkat lunak ini. Selain itu , agar dapat *multicompany/multiclient*.
- ²⁷

```
-- Table: ba_business
-- DROP TABLE ba_business;

CREATE TABLE ba_business
(
    id serial NOT NULL,
    create_uid integer, -- Created by
    create_date timestamp without time zone, -- Created on
    name character varying(200) NOT NULL, -- Business Name
    write_uid integer, -- Last Updated by
    write_date timestamp without time zone, -- Last Updated on
    CONSTRAINT ba_business_pkey PRIMARY KEY (id),
    CONSTRAINT ba_business_create_uid_fkey FOREIGN KEY (create_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_business_write_uid_fkey FOREIGN KEY (write_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL
)
WITH (
    OIDS=FALSE
);
ALTER TABLE ba_business
    OWNER TO openpg;
COMMENT ON TABLE ba_business
    IS 'Client Businesses';
COMMENT ON COLUMN ba_business.create_uid IS 'Created by';
COMMENT ON COLUMN ba_business.create_date IS 'Created on';
COMMENT ON COLUMN ba_business.name IS 'Business Name';
COMMENT ON COLUMN ba_business.write_uid IS 'Last Updated by';
COMMENT ON COLUMN ba_business.write_date IS 'Last Updated on';
```

Gambar 5.1: Implementasi tabel ba_business

- 1 • Tabel ba_etl_header
- 2 Tabel ini mencatat proses ETL yang ada di bawah perusahaan/bisnis yang ada.

```
CREATE TABLE ba_etl_header
(
    id serial NOT NULL,
    create_date timestamp without time zone, -- Created on
    name character varying(100) NOT NULL, -- Name
    create_uid integer, -- Created by
    is_active boolean, -- Active?
    business_id integer NOT NULL, -- Business
    write_uid integer, -- Last Updated by
    write_date timestamp without time zone, -- Last Updated on
    CONSTRAINT ba_etl_header_pkey PRIMARY KEY (id),
    CONSTRAINT ba_etl_header_business_id_fkey FOREIGN KEY (business_id)
        REFERENCES ba_business (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_header_create_uid_fkey FOREIGN KEY (create_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_header_write_uid_fkey FOREIGN KEY (write_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL
)
WITH (
    OIDS=FALSE
);
ALTER TABLE ba_etl_header
    OWNER TO openpg;
COMMENT ON TABLE ba_etl_header
    IS 'Store ETL process settings';
COMMENT ON COLUMN ba_etl_header.create_date IS 'Created on';
COMMENT ON COLUMN ba_etl_header.name IS 'Name';
COMMENT ON COLUMN ba_etl_header.create_uid IS 'Created by';
COMMENT ON COLUMN ba_etl_header.is_active IS 'Active?';
COMMENT ON COLUMN ba_etl_header.business_id IS 'Business';
COMMENT ON COLUMN ba_etl_header.write_uid IS 'Last Updated by';
COMMENT ON COLUMN ba_etl_header.write_date IS 'Last Updated on';
```

Gambar 5.2: Implementasi tabel ba_etl_header

- 3 • Tabel ba_etl_detail
- 4 Tabel ini mencata detail dari setiap proses ETL dibawah perusahaan/bisnis.

```

CREATE TABLE ba_etl_detail
(
    id serial NOT NULL,
    read_data_csv_auto_read boolean, -- Auto-read from a folder?
    sequence integer, -- Step
    write_uid integer, -- Last Updated by
    save_data_model_id integer, -- Target Data Model
    write_date timestamp without time zone, -- Last Updated on
    header_id integer, -- Process
    create_date timestamp without time zone, -- Created on
    create_uid integer, -- Created by
    name character varying(255) NOT NULL, -- Description
    read_data_csv_trial_file bytea, -- Upload Sample File
    read_data_csv_delimiter character varying, -- Delimiter
    process_type character varying NOT NULL, -- Process Type
    read_data_source_type character varying, -- Source Type
    read_data_csv_auto_read_path character varying, -- Auto-read Folder Path
    read_data_csv_is_header_present boolean, -- CSV Has Header Row?
    script_text text, -- Enter Script
    lookup_from character varying, -- Lookup From
    lookup_source_key_column_moved0 character varying, -- Source Data Key
    lookup_model_key_column_moved0 character varying, -- Table Lookup Column
    lookup_target_key_column character varying, -- Target Data Key
    lookup_model_value_column_moved0 character varying, -- Table Value Column
    lookup_model_id integer, -- Table
    merge_csv_delimiter_2 character varying, -- Source 2 Delimiter
    merge_csv_delimiter_1 character varying, -- Source 1 Delimiter
    merge_csv_trial_file_1 bytea, -- Source 1 Sample
    lookup_model_value_column integer, -- Table Value Column
    merge_csv_trial_file_2 bytea, -- Source 2 Sample
    lookup_model_key_column integer, -- Table Lookup Column
    merge_csv_is_header_present_1 boolean, -- Source 1 Has Header Row?
    merge_csv_is_header_present_2 boolean, -- Source 2 Has Header Row?
    merge_source_type_1 character varying, -- Source 1 Type
    merge_type character varying, -- Merge Type
    merge_source_type_2 character varying, -- Source 2 Type
)

```

Gambar 5.3: Implementasi tabel ba_etl_detail

```

merge_source_type_2 character varying, -- Source 2 Type
lookup_source_key_column integer, -- Source Data Key
read_data_model_id integer, -- Existing Model
CONSTRAINT ba_etl_detail_pkey PRIMARY KEY (id),
CONSTRAINT ba_etl_detail_create_uid_fkey FOREIGN KEY (create_uid)
    REFERENCES res_users (id) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE SET NULL,
CONSTRAINT ba_etl_detail_header_id_fkey FOREIGN KEY (header_id)
    REFERENCES ba_etl_header (id) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE SET NULL,
CONSTRAINT ba_etl_detail_lookup_model_id_fkey FOREIGN KEY (lookup_model_id)
    REFERENCES ir_model (id) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE SET NULL,
CONSTRAINT ba_etl_detail_lookup_model_key_column_fkey FOREIGN KEY (lookup_model_key_column)
    REFERENCES ir_model_fields (id) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE SET NULL,
CONSTRAINT ba_etl_detail_lookup_model_value_column_fkey FOREIGN KEY (lookup_model_value_column)
    REFERENCES ir_model_fields (id) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE SET NULL,
CONSTRAINT ba_etl_detail_lookup_source_key_column_fkey FOREIGN KEY (lookup_source_key_column)
    REFERENCES ba_etl_columns (id) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE SET NULL,
CONSTRAINT ba_etl_detail_read_data_model_id_fkey FOREIGN KEY (read_data_model_id)
    REFERENCES ir_model (id) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE SET NULL,
CONSTRAINT ba_etl_detail_save_data_model_id_fkey FOREIGN KEY (save_data_model_id)
    REFERENCES ir_model (id) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE SET NULL,
CONSTRAINT ba_etl_detail_write_uid_fkey FOREIGN KEY (write_uid)
    REFERENCES res_users (id) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE SET NULL
);
WITH (
    OIDS=FALSE
);

```

Gambar 5.4: Implementasi tabel ba_etl_detail2

```

ALTER TABLE ba_etl_detail
OWNER TO openpgp;
COMMENT ON TABLE ba_etl_detail
IS 'Store ETL method process details, based on process type';
COMMENT ON COLUMN ba_etl_detail.read_data_csv_auto_read IS 'Auto-read from a folder?';
COMMENT ON COLUMN ba_etl_detail.sequence IS 'Step';
COMMENT ON COLUMN ba_etl_detail.write_uid IS 'Last Updated by';
COMMENT ON COLUMN ba_etl_detail.save_data_model_id IS 'Target Data Model';
COMMENT ON COLUMN ba_etl_detail.write_date IS 'Last Updated on';
COMMENT ON COLUMN ba_etl_detail.header_id IS 'Process';
COMMENT ON COLUMN ba_etl_detail.create_date IS 'Created on';
COMMENT ON COLUMN ba_etl_detail.create_uid IS 'Created by';
COMMENT ON COLUMN ba_etl_detail.name IS 'Description';
COMMENT ON COLUMN ba_etl_detail.read_data_csv_trial_file IS 'Upload Sample File';
COMMENT ON COLUMN ba_etl_detail.read_data_csv_delimiter IS 'Delimiter';
COMMENT ON COLUMN ba_etl_detail.process_type IS 'Process Type';
COMMENT ON COLUMN ba_etl_detail.read_data_source_type IS 'Source Type';
COMMENT ON COLUMN ba_etl_detail.read_data_csv_auto_read_path IS 'Auto-read Folder Path';
COMMENT ON COLUMN ba_etl_detail.read_data_csv_is_header_present IS 'CSV Has Header Row?';
COMMENT ON COLUMN ba_etl_detail.script_text IS 'Enter Script';
COMMENT ON COLUMN ba_etl_detail.lookup_from IS 'Lookup From';
COMMENT ON COLUMN ba_etl_detail.lookup_source_key_column_moved0 IS 'Source Data Key';
COMMENT ON COLUMN ba_etl_detail.lookup_model_key_column_moved0 IS 'Table Lookup Column';
COMMENT ON COLUMN ba_etl_detail.lookup_target_key_column IS 'Target Data Key';
COMMENT ON COLUMN ba_etl_detail.lookup_model_value_column_moved0 IS 'Table Value Column';
COMMENT ON COLUMN ba_etl_detail.lookup_model_id IS 'Table';
COMMENT ON COLUMN ba_etl_detail.merge_csv_delimiter_2 IS 'Source 2 Delimiter';
COMMENT ON COLUMN ba_etl_detail.merge_csv_delimiter_1 IS 'Source 1 Delimiter';
COMMENT ON COLUMN ba_etl_detail.merge_csv_trial_file_1 IS 'Source 1 Sample';
COMMENT ON COLUMN ba_etl_detail.lookup_model_value_column IS 'Table Value Column';
COMMENT ON COLUMN ba_etl_detail.merge_csv_trial_file_2 IS 'Source 2 Sample';
COMMENT ON COLUMN ba_etl_detail.lookup_model_key_column IS 'Table Lookup Column';
COMMENT ON COLUMN ba_etl_detail.merge_csv_is_header_present_1 IS 'Source 1 Has Header Row?';
COMMENT ON COLUMN ba_etl_detail.merge_csv_is_header_present_2 IS 'Source 2 Has Header Row?';
COMMENT ON COLUMN ba_etl_detail.merge_source_type_1 IS 'Source 1 Type';

```

Gambar 5.5: Implementasi tabel ba_etl_detail3

1 ● Tabel basisdata ba_etl_columns

2 Tabel ini berfungsi menyimpan kolom-kolom yang terlibat dalam proses ETL, baik
3 dari sumber data maupun yang ditambahkan di tengah proses

```

CREATE TABLE ba_etl_columns
(
    id serial NOT NULL,
    create_uid integer, -- Created by
    create_date timestamp without time zone, -- Created on
    name character varying, -- Column Key
    write_uid integer, -- Last Updated by
    write_date timestamp without time zone, -- Last Updated on
    header_id integer, -- Process
    CONSTRAINT ba_etl_columns_pkey PRIMARY KEY (id),
    CONSTRAINT ba_etl_columns_create_uid_fkey FOREIGN KEY (create_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_columns_header_id_fkey FOREIGN KEY (header_id)
        REFERENCES ba_etl_header (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_columns_write_uid_fkey FOREIGN KEY (write_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL
)
WITH (
    OIDS=FALSE
);
ALTER TABLE ba_etl_columns
    OWNER TO openpg;
COMMENT ON TABLE ba_etl_columns
    IS 'Columns of source data tp be used in ETL process';
COMMENT ON COLUMN ba_etl_columns.create_uid IS 'Created by';
COMMENT ON COLUMN ba_etl_columns.create_date IS 'Created on';
COMMENT ON COLUMN ba_etl_columns.name IS 'Column Key';
COMMENT ON COLUMN ba_etl_columns.write_uid IS 'Last Updated by';
COMMENT ON COLUMN ba_etl_columns.write_date IS 'Last Updated on';
COMMENT ON COLUMN ba_etl_columns.header_id IS 'Process';

```

Gambar 5.6: Implementasi tabel ba_etl_columns

4 ● Tabel basisdata ba_etl_columns_mapping

5 Tabel ini menyimpan detail *mapping* kolom dari data *input* ke *output*.

```

CREATE TABLE ba_etl_columns_mapping
(
    id serial NOT NULL,
    source_column character varying(255), -- Source Column
    create_date timestamp without time zone, -- Created on
    create_uid integer, -- Created by
    destination_column character varying, -- Destination Column
    write_uid integer, -- Last Updated by
    write_date timestamp without time zone, -- Last Updated on
    header_id integer, -- Header
    CONSTRAINT ba_etl_columns_mapping_pkey PRIMARY KEY (id),
    CONSTRAINT ba_etl_columns_mapping_create_uid_fkey FOREIGN KEY (create_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_columns_mapping_header_id_fkey FOREIGN KEY (header_id)
        REFERENCES ba_etl_header (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_columns_mapping_write_uid_fkey FOREIGN KEY (write_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL
),
WITH (
    OIDS=FALSE
);
ALTER TABLE ba_etl_columns_mapping
    OWNER TO openpg;
COMMENT ON TABLE ba_etl_columns_mapping
    IS 'Columns mapping between process steps';
COMMENT ON COLUMN ba_etl_columns_mapping.source_column IS 'Source Column';
COMMENT ON COLUMN ba_etl_columns_mapping.create_date IS 'Created on';
COMMENT ON COLUMN ba_etl_columns_mapping.create_uid IS 'Created by';
COMMENT ON COLUMN ba_etl_columns_mapping.destination_column IS 'Destination Column';
COMMENT ON COLUMN ba_etl_columns_mapping.write_uid IS 'Last Updated by';
COMMENT ON COLUMN ba_etl_columns_mapping.write_date IS 'Last Updated on';
COMMENT ON COLUMN ba_etl_columns_mapping.header_id IS 'Header';

```

Gambar 5.7: Implementasi tabel ba_etl_columns_mapping

6 ● Tabel basisdata ba_etl_format_data_lines

7 Tabel ini menyimpan detail informasi dari setiap kolom khusus tipe ETL Format Data.

```

CREATE TABLE ba_etl_columns_mapping
(
    id serial NOT NULL,
    source_column character varying(255), -- Source Column
    create_date timestamp without time zone, -- Created on
    create_uid integer, -- Created by
    destination_column character varying, -- Destination Column
    write_uid integer, -- Last Updated by
    write_date timestamp without time zone, -- Last Updated on
    header_id integer, -- Header
    CONSTRAINT ba_etl_columns_mapping_pkey PRIMARY KEY (id),
    CONSTRAINT ba_etl_columns_mapping_create_uid_fkey FOREIGN KEY (create_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_columns_mapping_header_id_fkey FOREIGN KEY (header_id)
        REFERENCES ba_etl_detail (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_columns_mapping_write_uid_fkey FOREIGN KEY (write_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL
),
WITH (
    OIDS=FALSE
);
ALTER TABLE ba_etl_columns_mapping
    OWNER TO openpg;
COMMENT ON TABLE ba_etl_columns_mapping
    IS 'Columns mapping between process steps';
COMMENT ON COLUMN ba_etl_columns_mapping.source_column IS 'Source Column';
COMMENT ON COLUMN ba_etl_columns_mapping.create_date IS 'Created on';
COMMENT ON COLUMN ba_etl_columns_mapping.create_uid IS 'Created by';
COMMENT ON COLUMN ba_etl_columns_mapping.destination_column IS 'Destination Column';
COMMENT ON COLUMN ba_etl_columns_mapping.write_uid IS 'Last Updated by';
COMMENT ON COLUMN ba_etl_columns_mapping.write_date IS 'Last Updated on';
COMMENT ON COLUMN ba_etl_columns_mapping.header_id IS 'Header';

```

Gambar 5.8: Implementasi tabel ba_etl_columns_mapping

- 1 • Tabel basisdata ba_etl_sort_data_columns
- 2 Tabel ini menyimpan detail kolom dan arah sorting khusus tipe ETL *Sort Data*

```

CREATE TABLE ba_etl_sort_data_columns
(
    id serial NOT NULL,
    create_uid integer, -- Created by
    create_date timestamp without time zone, -- Created on
    sequence integer, -- Sort Order
    write_uid integer, -- Last Updated by
    column_key_moved0 character varying(255), -- Column Key
    sort_order character varying, -- Direction
    write_date timestamp without time zone, -- Last Updated on
    header_id integer, -- Header
    column_key integer NOT NULL, -- Column Key
    CONSTRAINT ba_etl_sort_data_columns_pkey PRIMARY KEY (id),
    CONSTRAINT ba_etl_sort_data_columns_column_key_fkey FOREIGN KEY (column_key)
        REFERENCES ba_etl_columns (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_sort_data_columns_create_uid_fkey FOREIGN KEY (create_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_sort_data_columns_header_id_fkey FOREIGN KEY (header_id)
        REFERENCES ba_etl_detail (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_sort_data_columns_write_uid_fkey FOREIGN KEY (write_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL
),
WITH (
    OIDS=FALSE
);
ALTER TABLE ba_etl_sort_data_columns
    OWNER TO openpg;
COMMENT ON TABLE ba_etl_sort_data_columns
    IS 'Details for "sort_data" process type';
COMMENT ON COLUMN ba_etl_sort_data_columns.create_uid IS 'Created by';
COMMENT ON COLUMN ba_etl_sort_data_columns.create_date IS 'Created on';
COMMENT ON COLUMN ba_etl_sort_data_columns.sequence IS 'Sort Order';
COMMENT ON COLUMN ba_etl_sort_data_columns.write_date IS 'Last Updated on';

```

Gambar 5.9: Implementasi tabel ba_etl_sort_data_columns

- 3 • Tabel basisdata ba_etl_derived_column_lines
- 4 Tabel ini menyimpan detail formula untuk menghasilkan *derived column* khusus tipe ETL *Derived Column*.

```

CREATE TABLE ba_etl_derived_column_lines
(
    id serial NOT NULL,
    function character varying, -- Formula
    create_date timestamp without time zone, -- Created on
    data_type character varying, -- Data Type
    write_uid integer, -- Last Updated by
    write_date timestamp without time zone, -- Last Updated on
    header_id integer, -- Header
    create_uid integer, -- Created by
    new_or_modify character varying(255), -- New Column or Modify
    length integer, -- Length
    column_key_moved0 character varying(255), -- Column Key
    parameter character varying(1000), -- Parameter
    column_key integer, -- Replace
    expression text, -- Expression
    new_column_key character varying(64), -- New Column
    CONSTRAINT ba_etl_derived_column_lines_pkey PRIMARY KEY (id),
    CONSTRAINT ba_etl_derived_column_lines_column_key_fkey FOREIGN KEY (column_key)
        REFERENCES ba_etl_columns (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_derived_column_lines_create_uid_fkey FOREIGN KEY (create_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_derived_column_lines_header_id_fkey FOREIGN KEY (header_id)
        REFERENCES ba_etl_detail (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_derived_column_lines_write_uid_fkey FOREIGN KEY (write_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL
),
WITH (
    OIDS=FALSE
);
ALTER TABLE ba_etl_derived_column_lines
OWNER TO openpg;

```

Gambar 5.10: Implementasi tabel ba_etl_derived_column_lines

1 ● Tabel basisdata ba_etl_lookup_fixed_lines

2 Tabel ini menyimpan pilihan *lookup* yang diambil dari himpunan pilihan *fixed*. Khusus
3 tipe ETL *lookup*.

```

CREATE TABLE ba_etl_lookup_fixed_lines
(
    id serial NOT NULL,
    create_uid integer, -- Created by
    create_date timestamp without time zone, -- Created on
    value character varying, -- Result Value
    write_uid integer, -- Last Updated by
    lookup character varying, -- Lookup Value
    write_date timestamp without time zone, -- Last Updated on
    header_id integer, -- Header
    CONSTRAINT ba_etl_lookup_fixed_lines_pkey PRIMARY KEY (id),
    CONSTRAINT ba_etl_lookup_fixed_lines_create_uid_fkey FOREIGN KEY (create_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_lookup_fixed_lines_header_id_fkey FOREIGN KEY (header_id)
        REFERENCES ba_etl_detail (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_lookup_fixed_lines_write_uid_fkey FOREIGN KEY (write_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL
),
WITH (
    OIDS=FALSE
);
ALTER TABLE ba_etl_lookup_fixed_lines
OWNER TO openpg;
COMMENT ON TABLE ba_etl_lookup_fixed_lines
IS 'Fixed lookup value set for lookup process';
COMMENT ON COLUMN ba_etl_lookup_fixed_lines.create_uid IS 'Created by';
COMMENT ON COLUMN ba_etl_lookup_fixed_lines.create_date IS 'Created on';
COMMENT ON COLUMN ba_etl_lookup_fixed_lines.value IS 'Result Value';
COMMENT ON COLUMN ba_etl_lookup_fixed_lines.write_uid IS 'Last Updated by';
COMMENT ON COLUMN ba_etl_lookup_fixed_lines.lookup IS 'Lookup Value';
COMMENT ON COLUMN ba_etl_lookup_fixed_lines.write_date IS 'Last Updated on';
COMMENT ON COLUMN ba_etl_lookup_fixed_lines.header_id IS 'Header';

```

Gambar 5.11: Implementasi tabel ba_etl_lookup_fixed_lines

4 ● Tabel basisdata ba_etl_save_data_columns

5 Tabel ini menyimpan *mapping* kolom antara hasil proses dengan *field* di tabel. Khusus
6 tipe ETL *Save Data*

```

CREATE TABLE ba_etl_save_data_columns
(
    id serial NOT NULL,
    source_column integer, -- Source Column
    create_date timestamp without time zone, -- Created on
    create_uid integer, -- Created by
    destination_column character varying(64), -- Destination Column
    write_uid integer, -- Last Updated by
    write_date timestamp without time zone, -- Last Updated on
    header_id integer, -- Header
    CONSTRAINT ba_etl_save_data_columns_pkey PRIMARY KEY (id),
    CONSTRAINT ba_etl_save_data_columns_create_uid_fkey FOREIGN KEY (create_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_save_data_columns_header_id_fkey FOREIGN KEY (header_id)
        REFERENCES ba_etl_detail (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_save_data_columns_source_column_fkey FOREIGN KEY (source_column)
        REFERENCES ba_etl_columns (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_save_data_columns_write_uid_fkey FOREIGN KEY (write_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL
)
WITH (
    OIDS=FALSE
);
ALTER TABLE ba_etl_save_data_columns
    OWNER TO openpg;
COMMENT ON TABLE ba_etl_save_data_columns
    IS 'Save data columns mapping';
COMMENT ON COLUMN ba_etl_save_data_columns.source_column IS 'Source Column';
COMMENT ON COLUMN ba_etl_save_data_columns.create_date IS 'Created on';
COMMENT ON COLUMN ba_etl_save_data_columns.create_uid IS 'Created by';
COMMENT ON COLUMN ba_etl_save_data_columns.destination_column IS 'Destination Column';
COMMENT ON COLUMN ba_etl_save_data_columns.write_uid IS 'Last Updated by';

```

Gambar 5.12: Implementasi tabel ba_etl_save_data_columns

1 • Tabel basisdata ba_etl_lookup_model_mappings

2 Tabel ini menyimpan *lookup* dari tabel lain, tabel ini menampung pasangan *field* di
3 data proses dengan yang di tabel lookup. Khusus tipe ETL *lookup*

```

CREATE TABLE ba_etl_lookup_model_mappings
(
    id serial NOT NULL,
    source_column integer, -- Lookup Source Column
    create_date timestamp without time zone, -- Created on
    create_uid integer, -- Created by
    write_uid integer, -- Last Updated by
    model_column integer, -- Lookup Table Column
    write_date timestamp without time zone, -- Last Updated on
    header_id integer, -- Header
    CONSTRAINT ba_etl_lookup_model_mappings_pkey PRIMARY KEY (id),
    CONSTRAINT ba_etl_lookup_model_mappings_create_uid_fkey FOREIGN KEY (create_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_lookup_model_mappings_header_id_fkey FOREIGN KEY (header_id)
        REFERENCES ba_etl_detail (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_lookup_model_mappings_model_column_fkey FOREIGN KEY (model_column)
        REFERENCES ir_model_fields (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_lookup_model_mappings_source_column_fkey FOREIGN KEY (source_column)
        REFERENCES ba_etl_columns (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_lookup_model_mappings_write_uid_fkey FOREIGN KEY (write_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL
)
WITH (
    OIDS=FALSE
);
ALTER TABLE ba_etl_lookup_model_mappings
    OWNER TO openpg;
COMMENT ON TABLE ba_etl_lookup_model_mappings
    IS 'Columns mapping between data source and target model/table';
COMMENT ON COLUMN ba_etl_lookup_model_mappings.source_column IS 'Lookup Source Column';
COMMENT ON COLUMN ba_etl_lookup_model_mappings.create_date IS 'Created on';

```

Gambar 5.13: Implementasi tabel ba_etl_lookup_model_mappings

4 • Tabel basisdata ba_etl_merge_columns1

5 Tabel ini menyimpan informasi *merge* dari sumber pertama. Khusus tipe ETL *Merge*

```

CREATE TABLE ba_etl_merge_columns1
(
    id serial NOT NULL,
    create_uid integer, -- Created by
    create_date timestamp without time zone, -- Created on
    is_included boolean, -- Output?
    join_field character varying, -- Join Field
    column_key character varying(64), -- Column Key
    write_date timestamp without time zone, -- Last Updated on
    header_id integer, -- Header
    write_uid integer, -- Last Updated by
    source_column character varying(64), -- Column Key
    destination_column character varying(64), -- Mapping
    CONSTRAINT ba_etl_merge_columns1_pk PRIMARY KEY (id),
    CONSTRAINT ba_etl_merge_columns1_create_uid_fkey FOREIGN KEY (create_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_merge_columns1_header_id_fkey FOREIGN KEY (header_id)
        REFERENCES ba_etl_detail (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_merge_columns1_write_uid_fkey FOREIGN KEY (write_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL
)
WITH (
    OIDS=FALSE
);
ALTER TABLE ba_etl_merge_columns1
    OWNER TO openpgp;
COMMENT ON TABLE ba_etl_merge_columns1
    IS 'Merge column settings for first data source';
COMMENT ON COLUMN ba_etl_merge_columns1.create_uid IS 'Created by';
COMMENT ON COLUMN ba_etl_merge_columns1.create_date IS 'Created on';
COMMENT ON COLUMN ba_etl_merge_columns1.is_included IS 'Output?';
COMMENT ON COLUMN ba_etl_merge_columns1.join_field IS 'Join Field';
COMMENT ON COLUMN ba_etl_merge_columns1.column_key IS 'Column Key';
COMMENT ON COLUMN ba_etl_merge_columns1.destination_column IS 'Mapping' ...

```

Gambar 5.14: Implementasi tabel ba_etl_merge_columns1

1 ● Tabel basisdata ba_etl_merge_columns2

2 Tabel ini menyimpan informasi *merge* dari sumber kedua. Khusus tipe ETL *Merge*

```

CREATE TABLE ba_etl_merge_columns2
(
    id serial NOT NULL,
    create_uid integer, -- Created by
    create_date timestamp without time zone, -- Created on
    is_included boolean, -- Output?
    join_field character varying, -- Join Field
    column_key character varying(64), -- Column Key
    write_date timestamp without time zone, -- Last Updated on
    header_id integer, -- Header
    write_uid integer, -- Last Updated by
    source_column character varying(64), -- Column Key
    destination_column character varying(64), -- Mapping
    CONSTRAINT ba_etl_merge_columns2_pk PRIMARY KEY (id),
    CONSTRAINT ba_etl_merge_columns2_create_uid_fkey FOREIGN KEY (create_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_merge_columns2_header_id_fkey FOREIGN KEY (header_id)
        REFERENCES ba_etl_detail (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_merge_columns2_write_uid_fkey FOREIGN KEY (write_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL
)
WITH (
    OIDS=FALSE
);
ALTER TABLE ba_etl_merge_columns2
    OWNER TO openpgp;
COMMENT ON TABLE ba_etl_merge_columns2
    IS 'Merge column settings for second data source';
COMMENT ON COLUMN ba_etl_merge_columns2.create_uid IS 'Created by';
COMMENT ON COLUMN ba_etl_merge_columns2.create_date IS 'Created on';
COMMENT ON COLUMN ba_etl_merge_columns2.is_included IS 'Output?';
COMMENT ON COLUMN ba_etl_merge_columns2.join_field IS 'Join Field';
COMMENT ON COLUMN ba_etl_merge_columns2.column_key IS 'Column Key';

```

Gambar 5.15: Implementasi tabel ba_etl_merge_columns2

3 ● Tabel basisdata ba_etl_aggregate_columns

4 Tabel ini menyimpan detail informasi proses aggregate khusus tipe ETL *aggregate*.

```

CREATE TABLE ba_etl_aggregate_columns
(
    id serial NOT NULL,
    create_uid integer, -- Created by
    create_date timestamp without time zone, -- Created on
    sequence integer, -- Sequence
    write_uid integer, -- Last Updated by
    column_key integer, -- Column Key
    write_date timestamp without time zone, -- Last Updated on
    header_id integer, -- Header
    operation character varying, -- Operation
    CONSTRAINT ba_etl_aggregate_columns_pkey PRIMARY KEY (id),
    CONSTRAINT ba_etl_aggregate_columns_column_key_fkey FOREIGN KEY (column_key)
        REFERENCES ba_etl_columns (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_aggregate_columns_create_uid_fkey FOREIGN KEY (create_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_aggregate_columns_header_id_fkey FOREIGN KEY (header_id)
        REFERENCES ba_etl_detail (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL,
    CONSTRAINT ba_etl_aggregate_columns_write_uid_fkey FOREIGN KEY (write_uid)
        REFERENCES res_users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE SET NULL
)
WITH (
    OIDS=FALSE
);
ALTER TABLE ba_etl_aggregate_columns
    OWNER TO opengp;
COMMENT ON TABLE ba_etl_aggregate_columns
    IS 'Definition details for aggregate process';
COMMENT ON COLUMN ba_etl_aggregate_columns.create_uid IS 'Created by';
COMMENT ON COLUMN ba_etl_aggregate_columns.create_date IS 'Created on';
COMMENT ON COLUMN ba_etl_aggregate_columns.sequence IS 'Sequence';
COMMENT ON COLUMN ba_etl_aggregate_columns.write_uid IS 'Last Updated by';

```

Gambar 5.16: Implementasi tabel ba_etl_aggregate_columns

5.3 Implementasi Kode Program Phyton

- 2 Berikut ini sisipan kode program perangkat lunak BI *tool*

- 3 1. Kode Program *load* kolom dari *source*

4

```

# buat load semua kolom dari source
def action_fillin_columns(self, cr, uid, ids, context={}):
    etl_id = ids[0]
    data = self.browse(cr, uid, etl_id, context=context)
    # kudu udah ada step minimal 1
    if len(data.etl_details) == 0:
        raise osv.except_osv_("ETL Error"), _('No load data process defined yet. Please define the process first.'))
    # step pertama harus load data atau merge
    step_info = data.etl_details[0]
    if step_info.process_type not in ['read_data','merge']:
        raise osv.except_osv_("ETL Error"), _('First process must be Read data or Merge.'))
    # kalau prosesnya read_data, baca aja langsung dari columns_mapping
    if step_info.process_type == 'read_data':
        etl_columns = []
        for mapping in step_info.columns_mapping:
            etl_columns.append(mapping.destination_column)
        if len(etl_columns) == 0:
            raise osv.except_osv_("ETL Error"), _('In the read data process, please define column mapping first.'))
    # kalau prosesnya merge, baca dari source 1 dan source 2, menskip yang tidak included dan yang dijadikan join column
    elif step_info.process_type == 'merge':
        etl_columns = []
        left_join_columns = []
        right_join_columns = []
        for column in step_info.merge_columns_1:
            if column.is_included: etl_columns.append(column.output_key)
            if column.join_field: left_join_columns.append(column.join_field)
        for column in step_info.merge_columns_2:
            if column.is_included: etl_columns.append(column.output_key)
            if column.join_field: right_join_columns.append(column.join_field)
        # hapus kolom2 yang dipakai untuk join. Kalau left atau inner join, hapus dari left_join_columns,
        # kalau right join hapus dari yang right_join_columns
        if step_info.merge_type in ['left','inner']:
            for column in left_join_columns:
                if column in etl_columns: etl_columns.remove(column)
        else:
            for column in right_join_columns:

```

Gambar 5.17: kode program *load columns from source*

- 5 2. Kode Program untuk *mapping column*

```

def _map_columns(self, source_data, columns_mapping, context={}):
    # kalau tidak ada columns_mapping, artinya hasil map idem aslinya
    if len(columns_mapping) == 0: return source_data
    result = []
    for row in source_data:
        result_row = {}
        for mapping in columns_mapping:
            if not mapping.destination_column: continue # kalau destination_
            # kalau source_column dalam bentuk integer, maka diasumsikan row ada
            # bila bukan maka diasumsikan row adalah sebuah dictionary
            is_key_idx = False
            try:
                key = mapping.source_column.name
            except:
                try:
                    key = int(mapping.source_column)
                    is_key_idx = True
                except:
                    key = mapping.source_column
            # tentukan data yang akan dimasukkan ke hasil mapping
            try:
                if is_key_idx:
                    cell_data = row[key]
                else:
                    cell_data = row.get(key, None)
            except:
                cell_data = None
            result_row.update((mapping.destination_column, cell_data))
        # beres ngurusin sebaris. tambahkan ke baris hasil
        result.append(result_row)
    # beres semua
    return result

def _etl_process_read_data(self, cr, uid, step_info, raw_data, context={}):
    # untuk read dari csv
    if step_info.read_data_source_type == 'csv':
        csv_data = step_info.read_data_csv_trial_file.decode('base64')

```

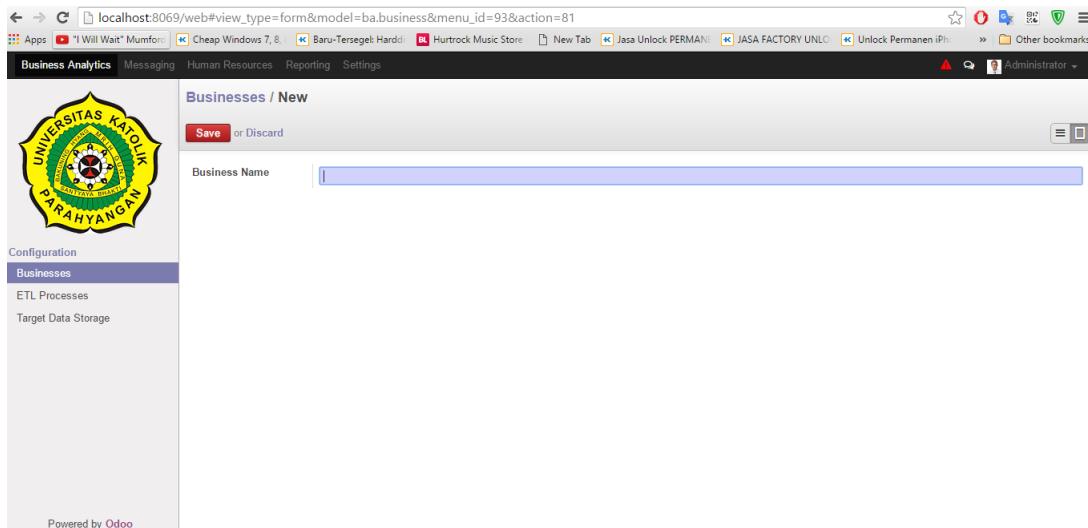
Gambar 5.18: Kode Program untuk *mapping column*

5.4 Implementasi Antar Muka

5.4.1 Implementasi Antar Muka untuk *Business*

- Buat Bisnis Baru

Halaman ini digunakan ketika pertama perusahaan akan membuat bisnis baru sebelum masuk pada proses ETL.

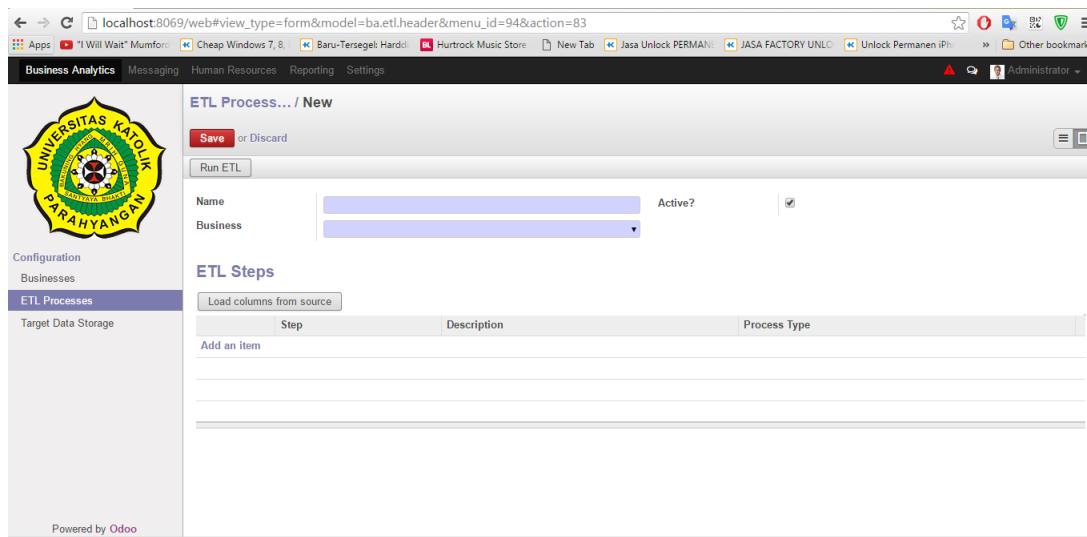


Gambar 5.19: Tampilan untuk membuat bisnis baru

5.4.2 Implementasi Antar Muka untuk Proses ETL

- Membuat Proses ETL Baru

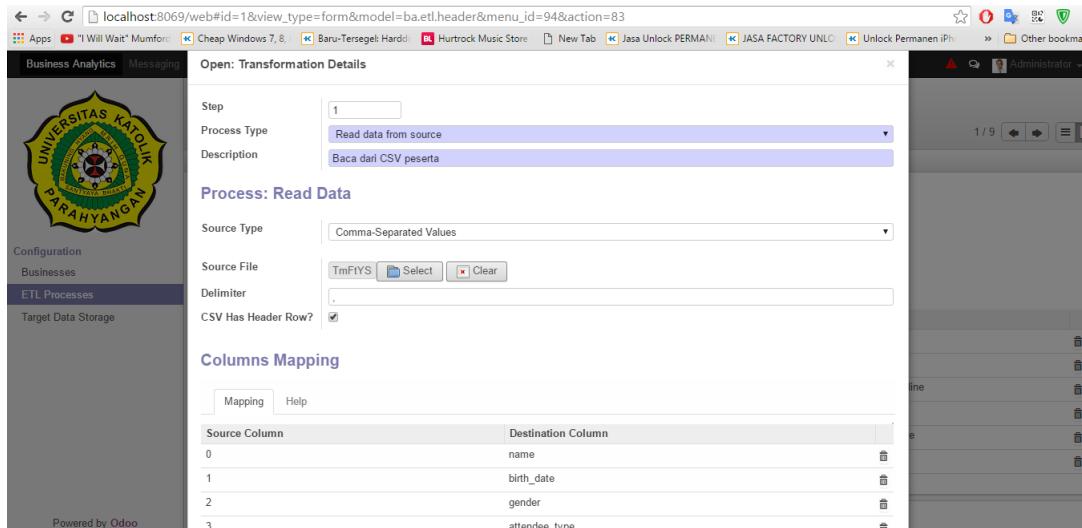
Halaman ini digunakan ketika membuat proses ETL baru.



Gambar 5.20: Tampilan untuk membuat ETL baru

1 • *Upload Source*

2 Halaman ini digunakan ketika pengguna meng-*upload data source*.

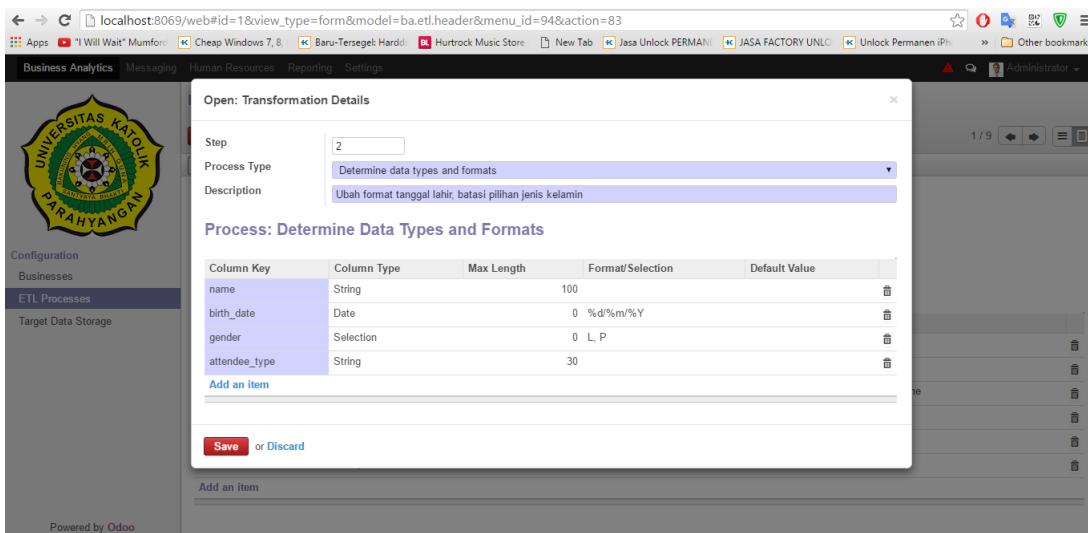


Gambar 5.21: Tampilan untuk *upload source*

3 • *Determine Data Types and Format*

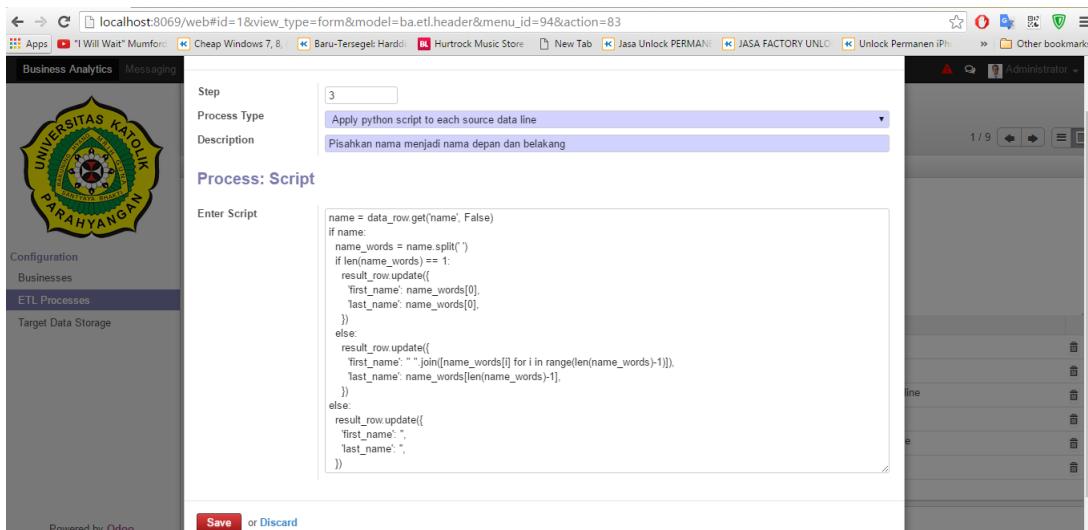
4 Halaman ini memudahkan pengguna untuk menentukan format dan tipe data pada *source*.

5



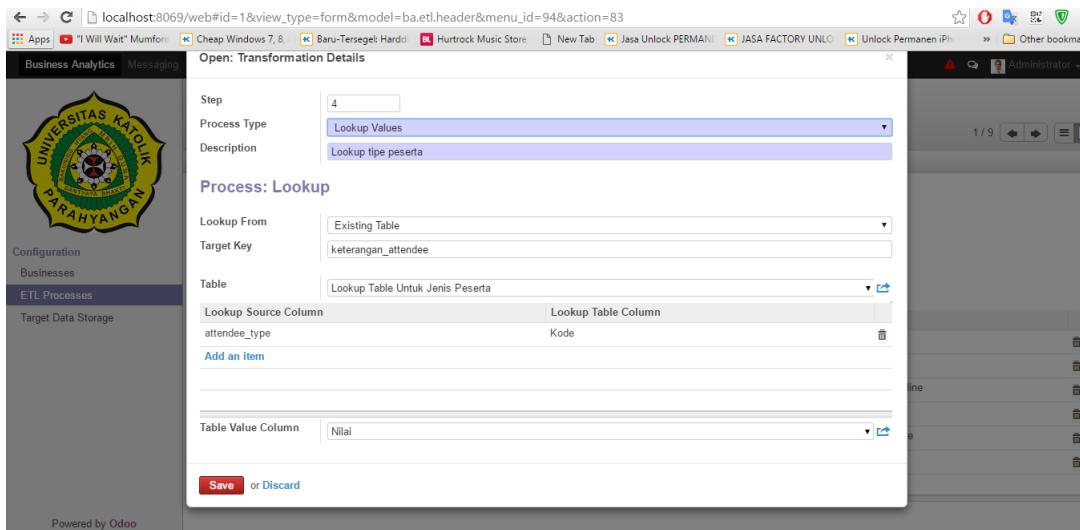
Gambar 5.22: Tampilan untuk ubah data dan format data

- 1 ● Menerapkan Skrip Phyton
- 2 Halaman ini dapat digunakan pengguna ketika akan menerapkan kode phyton untuk memodifikasi *source*.
- 3



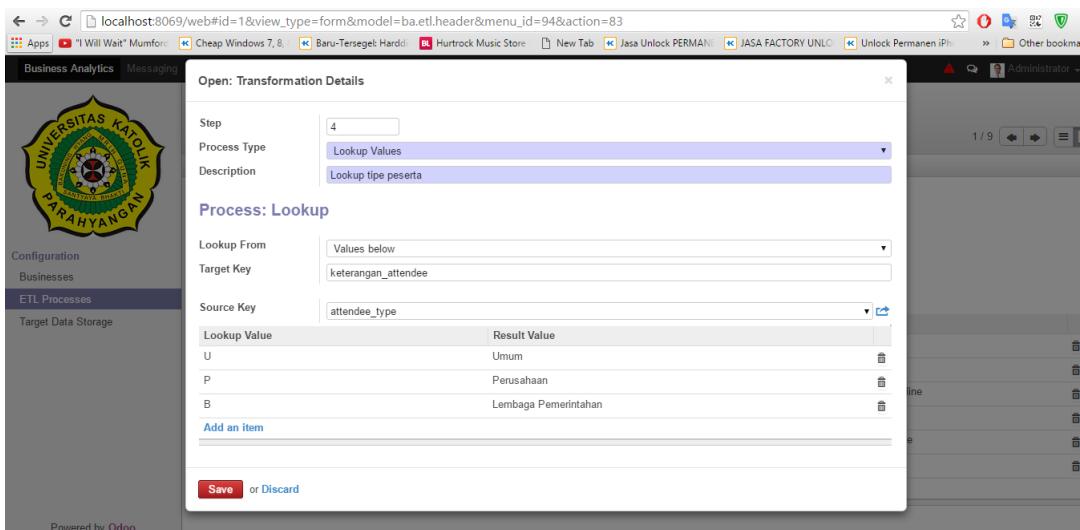
Gambar 5.23: Tampilan untuk menerapkan skrip phyton

- 4 ● Tampilan *Lookup* dengan *Existing Table* Halaman ini digunakan ketika pengguna ingin melakukan lookup dengan tabel lain yang sudah dimasukan terlebih dahulu dalam database
- 5
- 6



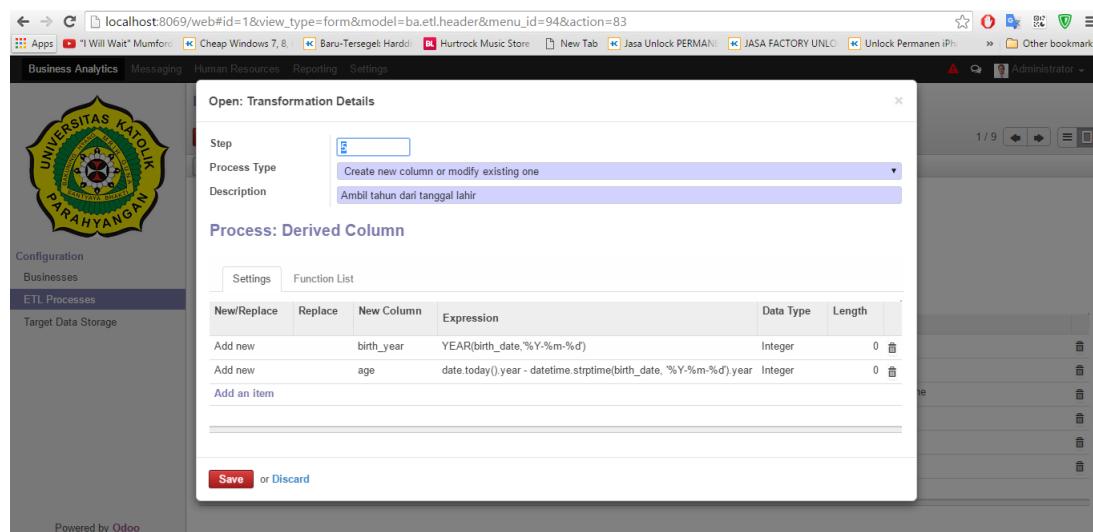
Gambar 5.24: Tampilan untuk *lookup* dari *existing table*

- 1 • Tampilan *Lookup* dengan *Values Below* Halaman ini digunakan ketika pengguna ingin melakukan lookup dengan nilai *lookup* dan hasil yang dimasukan langsung oleh pengguna
- 2
- 3



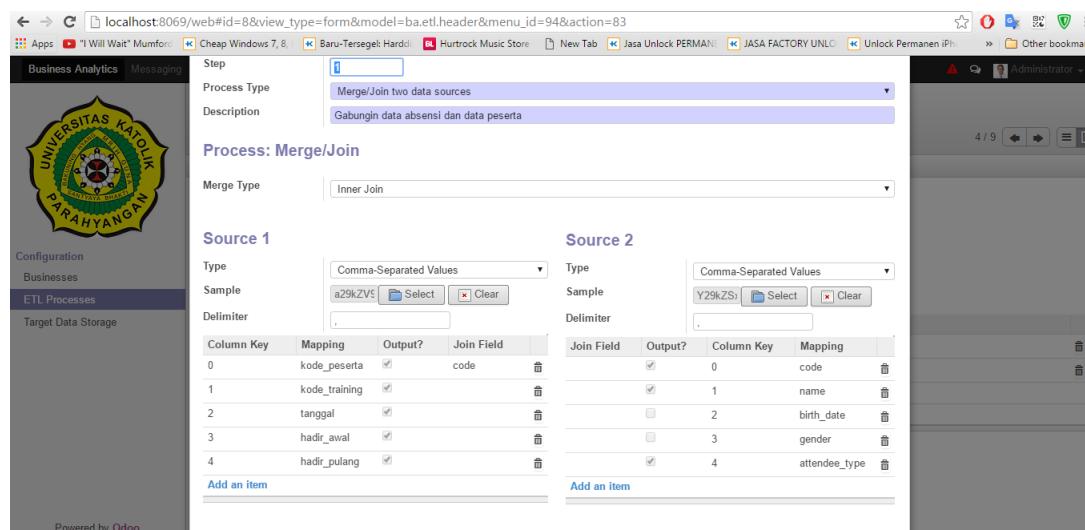
Gambar 5.25: Tampilan untuk *lookup* dari *value* yang dituliskan pengguna

- 4 • Menambah atau memodifikasi kolom yang tersedia
- 5 Halaman ini digunakan ketika pengguna ingin menambahkan kolom atau memodifikasi kolom yang tersedia.
- 6



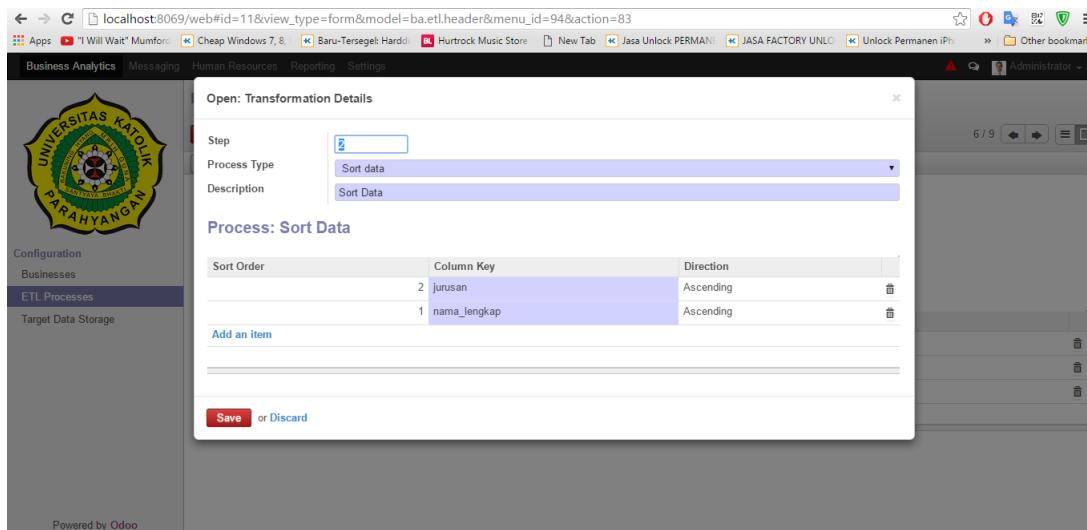
Gambar 5.26: Tampilan untuk menambah atau memodifikasi kolom

- 1 • *Merging* dua *data source*
- 2 Halaman ini digunakan ketika pengguna ingin menggabungkan dua *data source* menjadi satu.
- 3



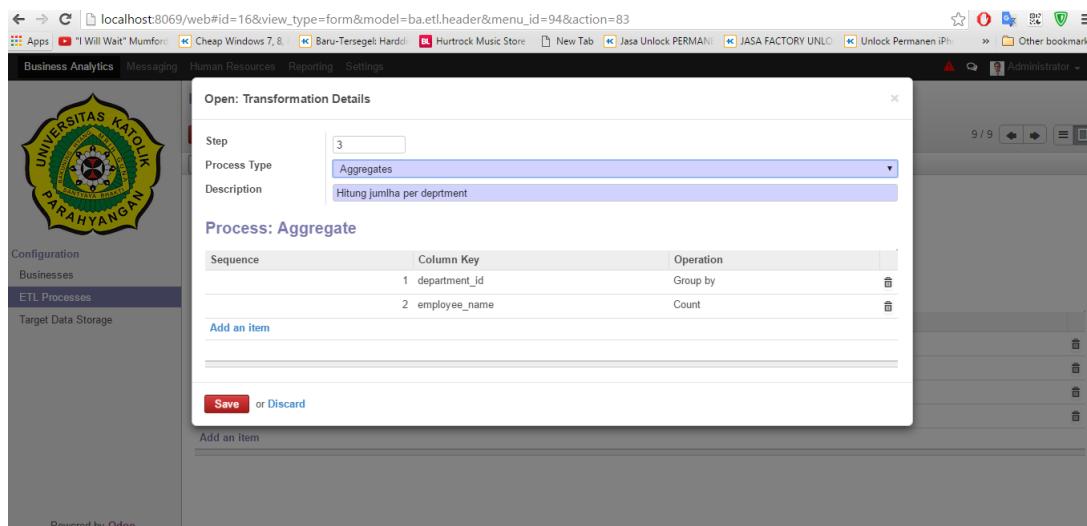
Gambar 5.27: Tampilan untuk *merging* dua *source*

- 4 • Melakukan *Sorting data*
- 5 Halaman ini digunakan ketika pengguna ingin melakukan *sorting data*.



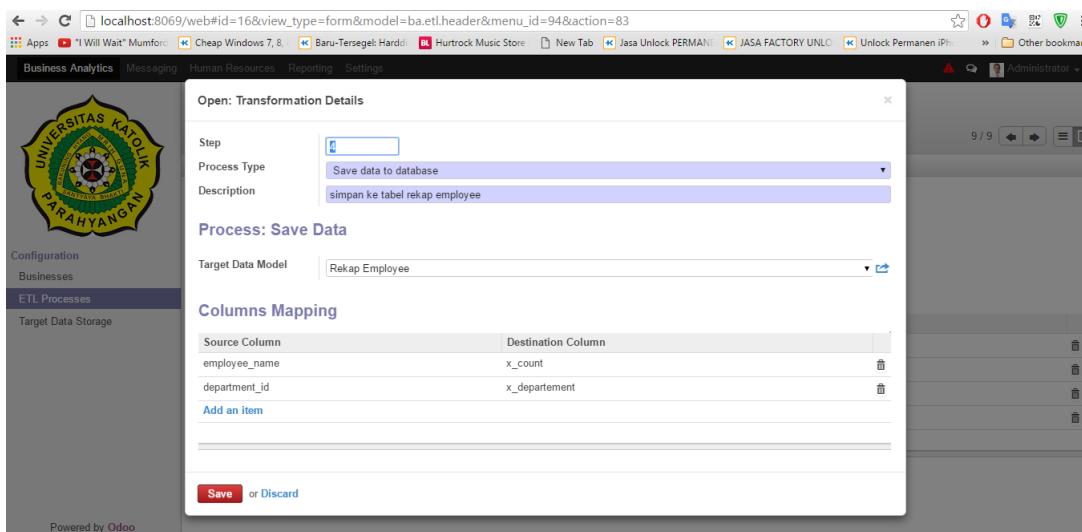
Gambar 5.28: Tampilan untuk melakukan *sort*

- 1 • Melakukan Agregat
- 2 Tampilan ini digunakan pengguna ketika akan melakukan agregat terhadap *source*



Gambar 5.29: Tampilan untuk melakukan agregat

- 3 • Menyimpan ke *database*
- 4 Halaman ini digunakan ketika pengguna ingin menyimpan hasil ETL kedalam *database*.



Gambar 5.30: Tampilan untuk menyimpan kedalam *database*

5.5 Pengujian

- 2 Pada subab ini akan dibahas mengenai pengujian terhadap perangkat lunak ini sehingga
3 dapat mengukur seberapa baik performansinya. berikut ini merupakan *sample data customer*
4 yang akan digunakan.

Tabel 5.1: Contoh Tabel *customer*

CIF	BRANCHCODE	GENDER	AGE	MARITAL STATUS	INCOME LEVEL	TENURE
1000000	ID0010096	2	15131	0	1	1175
1000043	ID0010075	2	14972	1	1	1175
1000044	ID0010049	2	13401	1	1	1175
1000056	ID0010049	1	18992	1	1	1175
1000059	ID0010038	2	20871	1	1	1175
1000064	ID0010049	2	20852	1	1	1175
1000067	ID0010038	1	15728	1	1	1175
1000068	ID0010049	1	16176	1	1	1175
1000069	ID0010026	1	19069	1	1	1175
1000075	ID0010038	1	11279	1	1	1175
1000076	ID0010090	1	18562	1	1	1175
1000084	ID0010090	2	14490	1	1	1175
1000140	ID0010129	1	15216	1	1	1174
1000149	ID0010026	1	9566	0	1	1174
1000166	ID0010108	2	14100	1	6	1174
1000195	ID0010014	1	12617	0	3	1174
1000202	ID0010132	1	23236	1	1	1174
1000244	ID0010096	2	19106	1	2	1174
1000246	ID0010038	2	27964	1	1	1174

- 5 Hasil Pengujian fitur adalah sebagai berikut.
- 6 1. *Sort, Aggregate, Modify Existing one, Read Source, Save to Database*
- 7 Hasil pengujian proses ETL dapat dilihat pada tabel dibawah ini.

Tabel 5.2: Tabel *ba_bussiness*

Hal yang diuji	Hasil yang diharapkan	Hasil Pengujian	Status
<i>Upload CSV</i>	Berhasil Melakukan <i>upload CSV</i>	Berhasil Melakukan <i>upload CSV</i>	OK
Melakukan Sorting	Data tersorting berdasarkan <i>branchcode</i> dan <i>gender</i> .	Data tersorting berdasarkan <i>branchcode</i> dan <i>gender</i>	OK
Melakukan Agregat	Hitung rata-rata umur per cabang per jenis kelamin	Data telah terhitung per rata-rata umur per cabang per jenis kelamin	OK
Memodifikasi data	Ubah umur ke dalam satuan tahun	Umur telah berhasil diubah kedalam satuan tahun	OK
Memodifikasi data	Ubah jenis kelamin menjadi P/L	Jenis kelamin menjadi P/L	OK
Menyimpan ke <i>database</i>	Simpan hasil ke <i>database</i>	Data berhasil tersimpan ke <i>database</i>	OK

BAB 6

KESIMPULAN DAN SARAN

6.1 Kesimpulan

Setelah melakukan penelitian maka dapat disimpulkan beberapa hal, yakni:

1. *Framework* OODOO mendukung untuk pembuatan ETL *tool* yang berperan dalam proses BI.
2. OODOO bersifat modular sehingga fleksibilitasnya tinggi.
3. Jika SSIS membutuhkan 2 *tool* untuk ETL yaitu visual studio dan management studio, OODOO hanya membutuhkan 1 tool saja.
4. Pengguna tanpa pengetahuan SQL dapat menggunakan *tool* ini.

6.2 Saran

Berikut ini saran yang diharapkan dapat membantu pengembangan penelitian ini lebih lanjut, yakni:

1. Menambahkan *input data* XML-RPC
2. Menambahkan OLAP dan Cube
3. *Merge* yang pada saat ini baru bisa di jalankan apabila ditempatkan paling pertama, diharapkan pada penelitian selanjutnya dapat ditempatkan dimana saja.

¹

DAFTAR REFERENSI

- ² [1] S. R. Matteo Golfarelli, *Data Warehouse Design: Modern Principles and Methodologies*.
³ Mc Graw Hill, 2009.
- ⁴ [2] Rasyid, "Pengertian mis, dss, data warehouse, data mining, olap, bi."
- ⁵ [3] C. Vercellis, *Business Intelligence: Data Mining and Optimization for Decision Making*.
⁶ WILEY, 2009.
- ⁷ [4] H. Lauri, pp. 37–39. 2014.

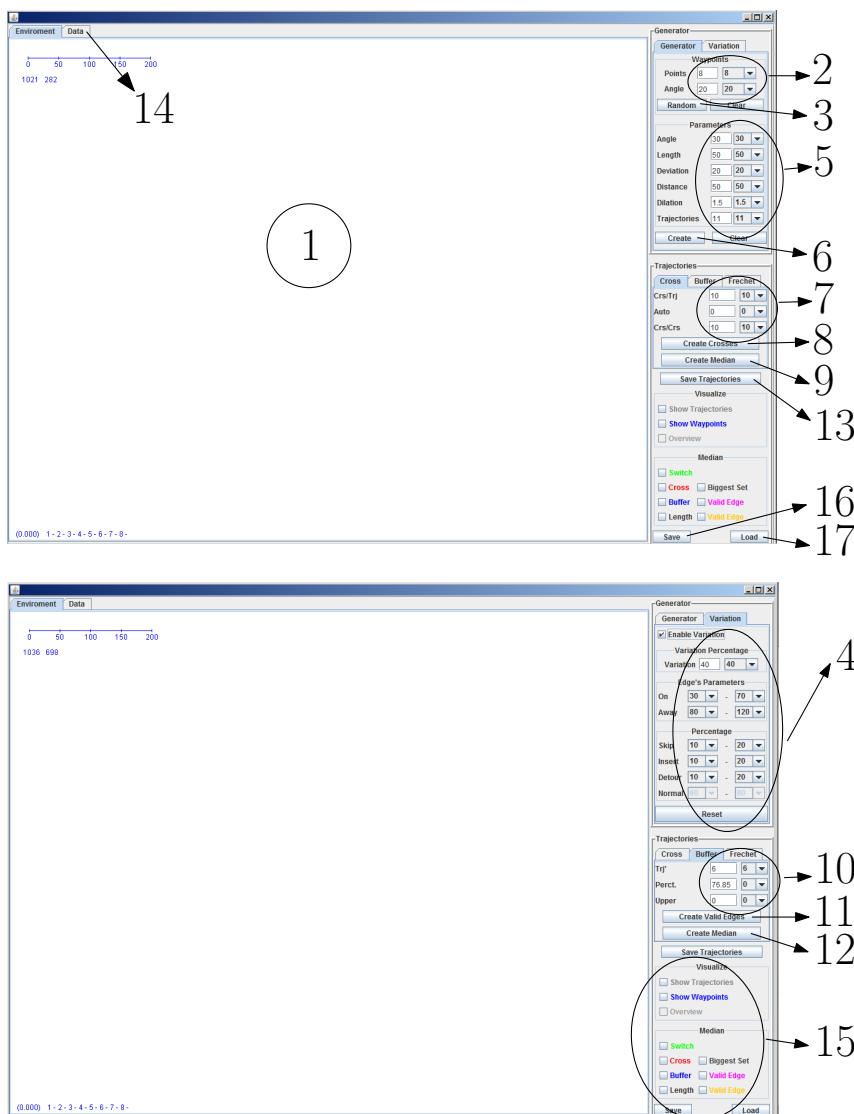
1

LAMPIRAN A

2

THE PROGRAM

3 The interface of the program is shown in Figure A.1:



Gambar A.1: Interface of the program

4

Step by step to compute the median trajectory using the program:

5

1. Create several waypoints. Click anywhere in the "Environment" area(1) or create them automatically by setting the parameters for waypoint(2) or clicking the button "Random"(3).

6

2. The "Variation" tab could be used to create variations by providing values needed to make them(4).

- 1 3. Create a set of trajectories by setting all parameters(5) and clicking the button “Cre-
2 ate”(6).
- 3 4. Compute the median using the homotopic algorithm:
 - 4 • Define all parameters needed for the homotopic algorithm(7).
 - 5 • Create crosses by clicking the “Create Crosses” button(8).
 - 6 • Compute the median by clicking the “Compute Median” button(9).
- 7 5. Compute the median using the switching method and the buffer algorithm:
 - 8 • Define all parameters needed for the buffer algorithm(10).
 - 9 • Create valid edges by clicking the “Create Valid Edges”button(11).
 - 10 • Compute the median by clicking the “Compute Median”button(12).
- 11 6. Save the resulting median by clicking the “Save Trajectories” button(13). The result
12 is saved in the computer memory and can be seen in “Data” tab(14)
- 13 7. The set of trajectories and its median trajectories will appear in the “Environment”
14 area(1) and the user can change what to display by selecting various choices in “Visu-
15 alize” and “Median” area(15).
- 16 8. To save all data to the disk, click the “Save”(16) button. A file dialog menu will appear.
- 17 9. To load data from the disk, click the “Load”(17) button.

1

LAMPIRAN B

2

THE SOURCE CODE

Listing B.1: MyFurSet.java

```

3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.HashSet;
6
7
8  /**
9   * 
10  * @author Lionov
11  */
12
13 // class for set of vertices close to furthest edge
14 public class MyFurSet {
15     protected int id;                                //id of the set
16     protected MyEdge FurthestEdge;                   //the furthest edge
17     protected HashSet<MyVertex> set;                //set of vertices close to furthest edge
18     protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each
19         trajectory
20     protected ArrayList<Integer> closeID;           //store the ID of all vertices
21     protected ArrayList<Double> closeDist;          //store the distance of all vertices
22     protected int totaltrj;                         //total trajectories in the set
23
24 /**
25  * Constructor
26  * @param id : id of the set
27  * @param totaltrj : total number of trajectories in the set
28  * @param FurthestEdge : the furthest edge
29  */
30 public MyFurSet(int id,int totaltrj ,MyEdge FurthestEdge) {
31     this.id = id;
32     this.totaltrj = totaltrj;
33     this.FurthestEdge = FurthestEdge;
34     set = new HashSet<MyVertex>();
35     ordered = new ArrayList<ArrayList<Integer>>();
36     for (int i=0;i<totaltrj ;i++) ordered.add(new ArrayList<Integer>());
37     closeID = new ArrayList<Integer>(totaltrj);
38     closeDist = new ArrayList<Double>(totaltrj);
39     for (int i = 0;i <totaltrj ;i++) {
40         closeID.add(-1);
41         closeDist.add(Double.MAX_VALUE);
42     }
43 }
44
45 /**
46  * set a vertex into the set
47  * @param v : vertex to be added to the set
48  */
49 public void add(MyVertex v) {
50     set.add(v);
51 }
52
53 /**
54  * check whether vertex v is a member of the set
55  * @param v : vertex to be checked
56  * @return true if v is a member of the set , false otherwise
57  */
58 public boolean contains(MyVertex v) {
59     return this.set.contains(v);
60 }
61 }
```


1

LAMPIRAN C

2

THE SOURCE CODE

Listing C.1: MyFurSet.java

```

3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.HashSet;
6
7
8  /**
9   * 
10  * @author Lionov
11  */
12
13 // class for set of vertices close to furthest edge
14 public class MyFurSet {
15     protected int id;                                //id of the set
16     protected MyEdge FurthestEdge;                   //the furthest edge
17     protected HashSet<MyVertex> set;                //set of vertices close to furthest edge
18     protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each
19         trajectory
20     protected ArrayList<Integer> closeID;           //store the ID of all vertices
21     protected ArrayList<Double> closeDist;          //store the distance of all vertices
22     protected int totaltrj;                         //total trajectories in the set
23
24 /**
25  * Constructor
26  * @param id : id of the set
27  * @param totaltrj : total number of trajectories in the set
28  * @param FurthestEdge : the furthest edge
29  */
30     public MyFurSet(int id,int totaltrj ,MyEdge FurthestEdge) {
31         this.id = id;
32         this.totaltrj = totaltrj;
33         this.FurthestEdge = FurthestEdge;
34         set = new HashSet<MyVertex>();
35         ordered = new ArrayList<ArrayList<Integer>>();
36         for (int i=0;i<totaltrj ;i++) ordered.add(new ArrayList<Integer>());
37         closeID = new ArrayList<Integer>(totaltrj);
38         closeDist = new ArrayList<Double>(totaltrj);
39         for (int i = 0;i <totaltrj ;i++) {
40             closeID.add(-1);
41             closeDist.add(Double.MAX_VALUE);
42         }
43     }
44
45 /**
46  * set a vertex into the set
47  * @param v : vertex to be added to the set
48  */
49     public void add(MyVertex v) {
50         set.add(v);
51     }
52
53 /**
54  * check whether vertex v is a member of the set
55  * @param v : vertex to be checked
56  * @return true if v is a member of the set , false otherwise
57  */
58     public boolean contains(MyVertex v) {
59         return this.set.contains(v);
60     }
61
62 /**
63  * create a column for table Gamma, sorted for each row
64  */
65     public void createColumn() {
66         for (MyVertex v : set) {
67             for (Integer key : v.vertexnum.keySet()) {
68                 for (Integer values : v.vertexnum.get(key)) {
69                     ordered.get(key).add(values);
70                 }
71             }
72         }
73         for (ArrayList<Integer> al : ordered) Collections.sort(al);
74     }
75
76 }
77 }
```