# *APPLICATION PROGRAMMING INTERFACE* (API) UNTUK *REVERSE ENGINEERING DATABASE* DARI *MULTI DATABASE MANAGEMENT SYSTEM (*DBMS) MENJADI *STURCTURED QUERY LANGUAGE* (SQL) *SCRIPT* DENGAN JAVA

**SKRIPSI**

**JURUSAN TEKNIK ELEKTRO**

Diajukan untuk memenuhi sebagian persyaratan
Memperoleh gelar Sarjana Teknik


**Disusun oleh**:

**NOURMA DEVITA P.**

**NIM. 0710630007 - 63**



**KEMENTERIAN PENDIDIKAN NASIONAL**

**UNIVERSITAS BRAWIJAYA**

**FAKULTAS TEKNIK**

**JURUSAN TEKNIK ELEKTRO**

**MALANG**


**2012**

**LEMBAR PERSETUJUAN**

***APPLICATION PROGRAMMING ERFACE* (API) UNTUK *REVERSE ENGINEERING DATABASE* DARI *MULTI DATABASE MANAGEMENT SYSTEM* (DBMS) MENJADI *STURCTURED QUERY LANGUAGE* (SQL) *SCRIPT* DENGAN JAVA**

**SKRIPSI**

**JURUSAN TEKNIK ELEKTRO**

Diajukan untuk memenuhi sebagian persyaratan
Memperoleh gelar Sarjana Teknik


**Oleh:**

**NOURMA DEVITA P.**
**NIM. 0710630007 - 63**

**Telah diperiksa dan disetujui oleh:**

| Dosen Pembimbing I | Dosen Pembimbing II |
|---|---|
| **Ir. Muhammad Aswin, MT.** | **Adharul Muttaqin, ST. MT** |
| **NIP.19640626 199002 1 001** | **NIP. 19760121 200501 1 001** |

# PENGANTAR

Alhamdulillah, segenap puji dan syukur penulis panjatkan kepada Allah SWT yang telah memberikan rahmat dan hidayahNya sehingga penulis dapat menyelesaikan skripsi dengan judul "***Application Programming Interface (*API*)* Untuk *Reverse Engineering Database* dari *Multi Database Management System (*DBMS*)* Menjadi *Sturctured Query Language* (SQL*) Script* dengan Java**", yang diajukan untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Teknik.

Kajian analisis pada tulisan ini berisi tentang bagaimana cara untuk melakukan *reverse engineering database* dari DBMS tertentu yaitu (MySQL, Oracle, dan PostgreSQl) sehingga menghasilkan suatu SQL *script* dengan susunan benar. Benar disini memiliki arti bahwa SQL *script* berisi *script* susunan tabel – tabel relasi yang berurutan dari tabel master hingga tabel yang mereferensi tabel lain, serta *script* komponen – komponen *database* lain yang memiliki struktur yang benar. Penyusunan SQL *script* secara berurutan seperti ini diharapkan dapat membantu dalam memahami lebih cepat suatu *database* yang di *reverse engineering*.

Tidak banyak yang bisa penulis sampaikan kecuali ungkapan terima kasih kepada berbagai pihak yang telah dengan tulus ikhlas memberikan bimbingan, arahan, dan dukungan hingga penulisan tugas akhir ini dapat terselesaikan. Pada kesempatan kali ini, penulis mengucapkan banyak terima kasih kepada:

1. Bapak Dr. Ir. Sholeh Hadi Pramono ,M.S. selaku Ketua Jurusan Teknik Elektro.
2. Bapak M. Azis Muslim, ST., MT., Ph.D selaku Sekretaris Jurusan Teknik Elektro.
3. Bapak Waru Djuriatno, ST. MT. selaku Ketua Kelompok Dosen Keahlian Teknik Rekayasa Komputer.
4. Bapak Ir. Muh. Aswin ,M.T. selaku dosen pembimbing skripsi, yang telah banyak memberikan dukungan dan koreksi kepada penulis.

5. Bapak Adharul Muttaqin, ST., MT. selaku dosen pembimbing skripsi, yang telah banyak memberikan dukungan dan koreksi kepada penulis.
6. Seluruh dosen dan karyawan Jurusan Teknik Elektro.
7. Ayahanda Nursigit, Ibunda Nurchayati serta keluarga besar yang telah memberikan banyak dukungan secara moral maupun materi.
8. Donny Kurniawan dan Maharani Firdausi yang selalu memberikan bantuan, arahan, dukungan, semangat, dan do'a hingga terselesaikannya Tugas Akhir ini.
9. Andik Nur Achmad, Asri Samsiar Ilmananda, Ferdiansyah G., Adelia R. S, Farasofi N. R., Yayuk I., dan Agung N. P., kawan kerabat seperjuangan yang selalu saling membantu dan saling memberikan dukungan serta do'a hingga terselesaikannya Tugas Akhir ini.
10. Teman-teman Core angkatan 2007.
11. Seluruh Asisten Laboratotium Informatika dan Komputer serta teman-teman RisTIE karena selalu memberikan bantuan dan dukungan dalam penyelesaian Tugas Akhir ini.
12. Semua pihak yang tidak dapat saya sebutkan satu persatu atas dukungannya dan bantuannya.

Penulis menyadari sepenuhnya bahwa Tugas Akhir ini masih banyak kekurangan. Segala kritik dan saran yang membangun akan penulis terima demi kesempurnaan skripsi ini. Harapan penulis semoga skripsi ini bermanfaat bagi pembaca dan khususnya mahasiswa jurusan teknik elektro dimasa yang akan datang.

Malang, Januari 2012

Penyusun

## DAFTAR ISI

## DAFTAR GAMBAR

## DAFTAR TABEL

## ABSTRAK

**Nourma Devita P.**, Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, Januari 2012, *Application Programming Interface (*API*)* Untuk *Reverse Engineering Database* dari *Multi Database Management System (*DBMS*)* Menjadi *Sturctured Query Language* (SQL*) Script* dengan Java, Dosen Ir. Muhammad Aswin, MT. dan Adharul Muttaqin, ST. MT.

*Database* adalah suatu kumpulan informasi yang disimpan secara sistematik sehingga mempermudah dalam pengelolaan dan pencariannya. Selain data sebagai informasi yang penting untuk dijaga, struktur dari *database* sendiri juga merupakan hal yang penting untuk diketahui. Dengan struktur penyusunan *database* yang tepat, maka data dapat disimpan serta dikelola dengan baik. Salah satu cara untuk mengetahui struktur dari *database* adalah dengan melakukan *reverse engineering database* menjadi SQL *script.*

Untuk melakukan *reverse engineering database* menjadi SQL *script*, hal utama yang dibutuhkan sebelum mendapatkan datanya adalah informasi tentang metadata dari *database*. Metadata dapat diartikan sebagai suatu "*data about data*" (atau data tentang data), yang menyediakan susunan serta uraian informasi tentang data yang sesungguhnya. Dari metadata, dapat disusun kumpulan SQL *script* yang membentuk keseluruhan struktur *database*. Metadata *database* diakses melalui *information_schema* yang ada pada DBMS (*Database Management System*) dan diambil dengan mengirimkan kombinasi *query* yang disesuaikan dengan struktur *information_schema*.

Sistem *reverse engineering database* menjadi SQL *script* yang dirancang dan dibuat ini adalah berupa *Application Programming Interface* (API) dalam Java. API. Perancangan dan implementasi API ini dikerjakan dengan beberapa tahap. Tahapan tersebut meliputi proses mendapatkan koneksi dari DBMS, proses pemeriksaan DBMS vendor, proses mendapatkan metadata, proses melakukan *sorting database metadata component*, proses *generate* SQL *script*, dan proses pembuatan *output* berisi SQL *script*. Pembuatan secara berkelompok dan bertahap tersebut bertujuan untuk memudahkan penganalisaan API pada setiap bagian maupun secara keseluruhan.

Pengujian API untuk *reverse engineering database* ini adalah dengan mengimplementasikan API dalam sebuah aplikasi *reverse engineering database* sederhana, sehingga aplikasi tersebut menjadi media pembukti bahwa API dapat melakukan *reverse engineering database* dengan menggunakan koneksi yang didapat, yaitu dari multi DBMS (MySQL, Oracle, PostgreSQL) menjadi SQL *script* dengan struktur *database* yang benar. Hasil pengujian *reverse engineering database* menunjukkan bahwa SQL *script* yang dihasilkan dari aplikasi adalah berupa *output* String yang ditampilkan pada *form output,* atau berupa *file* berekstensi sql. Pengujian eksekusi membuktikan bahwa SQL *script* hasil *reverse engineering* dapat dieksekusi kembali pada DBMS asal serta memiliki struktur yang sesuai dengan struktur asal.



**Kata Kunci :** Java API, *reverse engineering database,* SQL *script* metadata, MySQL, Oracle, PostgreSQL

# BAB I
# PENDAHULUAN

## 1.1 LATAR BELAKANG

*Database* adalah suatu kumpulan informasi yang disimpan secara sistematik sehingga mempermudah dalam pengelolaan dan pencariannya [MAT-05 : 4]. Informasi yang dimaksud adalah berupa data. Data merupakan fakta yang dapat berupa angka-angka, kata-kata, ataupun citra yang menunjukkan suatu situasi tertentu [ANO-11]. Dari pengertiannya, maka dapat disimpulkan bahwa data merupakan sesuatu yang penting untuk dijaga keasliannya dan harus diorganisasikan sebaik – baiknya. Dalam konteks sistem aplikasi komputer, data penting akan diolah dan disimpan dalam suatu media penyimpanan dengan dibantu oleh *Database Management System* (DBMS). DBMS adalah sekumpulan program komputer yang berfungsi untuk mengontrol penyimpanan, pengelolaan, dan pengambilan data dari suatu *database* [OKE-09 : 2].

Salah satu fungsi DBMS yang penting adalah fungsi untuk melakukan *reverse engineering database* menjadi SQL *script.* Untuk melakukan *reverse engineering database* menjadi SQL *script*, hal utama yang dibutuhkan sebelum mendapatkan datanya adalah informasi tentang metadatanya. Dari metadata itulah dapat disusun kumpulan *script* yang membentuk sebuah *file* SQL *script* yang siap dieksekusi atau siap untuk diperbaiki kembali.

Pada beberapa DBMS, untuk melakukan *reverse engineering database* menjadi SQL *script*, dapat dilakukan dengan program tertentu. Program untuk melakukan eksport seperti ini akan berbeda untuk setiap DBMS yang digunakan. Java dengan teknologi Java yang terkait, seperti JDBC dan JDBC *Driver*, merupakan salah satu bahasa pemrograman yang menyediakan *tools* yang diperlukan untuk pengembangan antarmuka yang berhubungan dengan *database*. Jadi, adalah mungkin untuk membuat sebuah antarmuka *database* yang mampu menghubungkan dan melakukan *query database* dengan menggunakan bahasa yang berorientasi objek ini [BUR-XX : 1]. Berdasarkan alasan tersebut, skripsi ini bermaksud untuk mengetahui bagaimana cara untuk melakukan *reverse engineering database* sehingga menghasilkan suatu SQL *script* dengan susunan

yang benar. Benar disini memiliki arti bahwa SQL *script* berisi *script* susunan tabel – tabel relasi yang berurutan dari tabel master hingga tabel yang memiliki referensi, serta *script* komponen – komponen *database* yang lain. Penyusunan SQL *script* secara berurutan, diharapkan dapat membantu dalam memahami lebih cepat suatu *database* yang di *reverse engineer*. Hasil akhir yang ingin dicapai dari skripsi ini adalah membuat sebuah *Application Programming Interface (*API*)* yang berfungsi untuk melakukan *reverse engineering database* menjadi SQL *script* untuk beberapa DBMS yang sering digunakan (MySQL, Oracle, dan PostgreSQL) dengan menggunakan bahasa pemrograman Java.

## 1.2 RUMUSAN MASALAH

Berdasarkan uraian yang telah dikemukakan pada bagian latar belakang, maka dirumuskan beberapa permasalahan sebagai berikut :

1. Bagaimana merancang API Java yang dapat digunakan untuk melakukan *reverse engineering database* dari beberapa DBMS, yaitu MySQL, Oracle, dan PostgreSQL.
2. Bagaimana cara mendapatkan metadata dari beberapa DBMS yaitu MySQL, Oracle, dan PostgreSQL.
3. Bagaimana cara menyusun sebuah *file* SQL *script* yang benar dari informasi metadata yang didapatkan.
4. Bagaimana menguji API Java yang dapat digunakan untuk melakukan *reverse engineering database* dari beberapa DBMS, yaitu MySQL, Oracle, dan PostgreSQL.

## 1.3 BATASAN MASALAH

Dalam perencanaan dan pembuatan skripsi ini perlu diberikan pembatasan terhadap masalah. Batasan-batasan yang diberikan dalam penulisan skripsi ini adalah sebagai berikut:

1. *Aplication Programming Interface (*API*)* yang dibuat adalah menggunakan Java SE 6 dan JDBC 3.0.
2. Koneksi DBMS yang diterima adalah koneksi menggunakan JDBC.

3. *Database Management System* (DBMS) yang digunakan adalah MySQL 5.5.18, Oracle 10 *Express Edition*, dan PostgreSQL 9.0.3.
4. SQL *Script* yang akan di-*reverse* dari DBMS meliputi *script Data Definition Language* (DDL) untuk *base table* dan relasinya, *view*, *trigger*, *stored procedure*, *stored function*, *sequence* (Oracle, PostgreSQL) dan *script Data Manipulaton Language* (DML) untuk *insert data*.
5. Detil informasi metadata yang dapat diambil dan di-*reverse* dalam setiap tabel dalam *database* adalah informasi *table basic*, antara lain :
   a. Kolom dan atribut standar dari kolom (tipe data (*numeric*, *character*, *date / time*, *boolean*), *default value*, *auto increment* (MySQL), *null constraint*)
   b. *Index Key*
   c. *Primary Key*
   d. *Foreign Key*
   e. *Unique Key*
6. Pengujian dilakukan dengan membuat sebuah aplikasi desktop sederhana yang memanfaatkan API tersebut.

## 1.4 TUJUAN

Tujuan dari penulisan skripsi ini adalah merancang dan membuat sebuah *Application Programming Interface (*API*)* yang berfungsi untuk melakukan *reverse engineering database* menjadi SQL *script* untuk beberapa DBMS yang sering digunakan (MySQL, Oracle, dan PostgreSQL) dengan menggunakan bahasa pemrograman Java.

## 1.5 MANFAAT

Manfaat yang diharapkan dapat diperoleh melalui pengerjaan skripsi ini adalah :

- Bagi Penyusun :
  1. Penyusun dapat merancang dan membuat Java API yang dapat terkoneksi dengan *Multi* DBMS yang terdiri dari MySQL, Oracle, dan PostgreSQL sehingga dapat digunakan untuk melakukan *reverse engineering database*.
  2. Menerapkan ilmu yang telah diperoleh dari Teknik Elekto Konsentrasi Teknik Informatika dan Komputer Universitas Brawijaya.
  3. Menambah wawasan tentang beberapa macam DBMS serta memperoleh pemahaman lebih tentang perancangan dan pengembangan Java API.
- Bagi Pengguna :
  1. API dapat bermanfaat dalam pembuatan aplikasi java yang berkaitan dengan *reverse engineering database.*
  2. Aplikasi sederhana sebagai sarana pengujian dapat langsung digunakan untuk melakukan *reverse engineering database* dari DBMS yang telah ditentukan.

## 1.6 SISTEMATIKA PENULISAN

Sistematika penulisan dan gambaran untuk setiap bab pada laporan skripsi ini adalah sebagai berikut :

**BAB I Pendahuluan**

Menjelaskan latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika penulisan skripsi.

**BAB II Dasar Teori**

Menjelaskan kajian pustaka dan dasar teori yang digunakan.

**BAB III Metodologi**

Menjelaskan metode yang digunakan dalam pengerjaan skripsi.

**BAB IV Perancangan dan Implementasi**

Menjelaskan analisis perancangan perangkat lunak dan Implementasi alur diagram pembuatan API Java yang dapat digunakan untuk melakukan *reverse engineering database* dari beberapa DBMS, yaitu MySQL, Oracle, dan PostgreSQL.

**BAB V Pengujian**

Menjelaskan langkah-langkah pengujian dari sistem yang telah dibuat dan membahas hasil pengujiannya.

**BAB VI Kesimpulan dan Saran**

Berisi Kesimpulan dan Saran.

# BAB II
# DASAR TEORI

## 2.1 Database

*Database* adalah suatu kumpulan informasi yang disimpan secara sistematis sehingga mempermudah dalam pengelolaan dan pencariannya [MAT-05 : 4]. Sistematika dari suatu *database* berasal dari cara pengorganisasian data sesuai dengan model *database*-nya. Model *database* yang sering digunakan saat ini adalah model *relational database*.

Suatu *database* akan disimpan dalam media penyimpanan. Media penyimpanan tersebut dapat diibaratkan sebuah *storage* penyimpanan, misalnya *hardisk*. Dalam *database*, data yang ada tidak hanya diletakkan dan disimpan begitu saja dalam sebuah media penyimpanan, akan tetapi dikelola dengan sebuah sistem pengaturan *database* yang sering disebut dengan *Database Management System* (DBMS). Dengan begitu, suatu data dengan jumlah besar dan kompleks dapat tersusun sangat baik sehingga memungkinkan pengaksesan data dengan mudah dan cepat oleh pengguna [OKE-09 : 11].

## 2.2 Database Relasional

*Relational Database* adalah sebuah *database* dimana data dan *database* tersusun sesuai dengan aturan model relasional. Istilah "*relational database*", terkadang secara tidak formal digunakan untuk merujuk kepada *Relational Database Management System* (RDBMS). Istilah *relational database* awalnya didefinisikan dan diciptakan oleh Edgar Codd di Pusat Penelitian Almaden IBM 1970. Secara tepat, *relational database* adalah kumpulan dari sesuatu (biasanya disebut dengan tabel) yang saling berhubungan. Sedangkan item yang lain dianggap sebagai bagian dari *database* yang membantu dalam pengaturan data. Berikut merupakan istilah – istilah penting dari *relational database* yang setara dengan SQL *database* [OKE-09 : 19] :

| Common Term | DBMS Terminology | RDBMS Terminology |
|---|---|---|
| Database | Table | Database |
| Table | Table | Relation |
| Column | Field | Attribute |
| Row | Record | Tuple |

**Gambar 2.1** Istilah – Istilah dalam *Relational Database*

**Sumber :** Okereke, Gerald C., Eco Communication Inc., Lagos Ikeja. 2009

### 2.2.1 Tabel

Tabel didefinisikan sebagai satu set baris dan kolom. Baris, biasanya merepresentasikan suatu objek beserta informasi tentang objek tersebut. Objek biasanya berupa benda fisik atau konseptual. Setiap objek memiliki detil atribut yang dibagi dalam beberapa kolom. [OKE-09 : 19].

#### 2.2.1.1 Base dan Derived Table

Dalam sebuah *relational database* data disimpan dan diakses melalui tabel. Tabel – tabel yang menyimpan data disebut "*Base Tables*" dimana inplementasinya merupakan tabel-tabel. Komponen-komponen yang lain tidak menyimpan data, tetapi dengan menambahkan operasi relasional terhadap *base table*. Komponen seperti ini biasanya disebut dengan "*Derived Relations*" dimana implementasinya berupa *views* atau *queries*. Adanya *derived relations* sangatlah membantu dalam pengambilan informasi dari beberapa hubungan karena mereka bertindak sebagai *single relation* [OKE-09 : 20].

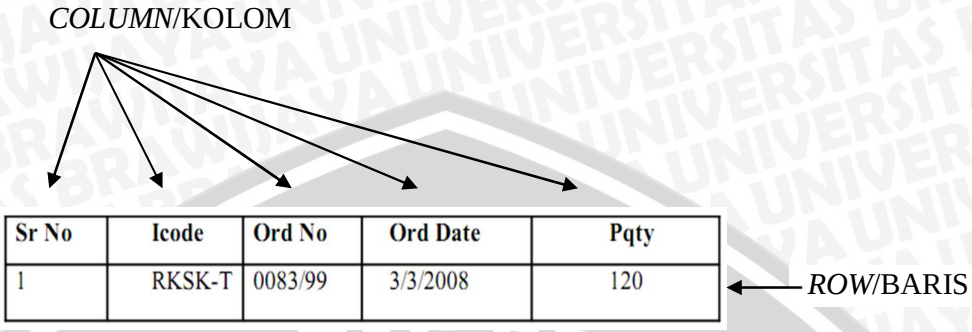
#### 2.2.1.2 *Row* / Baris

*Row* / baris merupakan sekumpulan data kolom yang saling berhubungan.

#### 2.2.1.3 *Column* / Kolom

*Field* / kolom merupakan suatu properti yang menyimpan bagian – bagian informasi dari satu entitas atau objek.

Sebagai contoh, Gambar 2.2 berikut menunjukkan hubungan antara baris dan kolom :




**Gambar 2.2 Tabel hubungan Baris dan Kolom**

**Sumber :** Okereke, Gerald C., Eco Communication Inc., Lagos Ikeja. 2009

### 2.2.1.4 *Index*

Tabel dalam suatu *database* dapat berkembang semakin besar, begitu pula dengan data yang disimpan di dalamnya. Akan tetapi, semakin besar tabel beserta isinya,maka untuk mengambil sebuah data tertentu akan semakin lama. Untuk menjaga performa dari *query database*, sangatlah penting untuk menambahkan *index* dalam suatu tabel. *Index* memudahkan dalam pencarian suatu nilai dalam kolom, sehingga pencarian berdasarkan *index* akan lebih cepat daripada tidak menggunakan *index* [DUB-06 : 8.6]. *Index* merupakan suatu mekanisme internal yang memudahkan dalam pencarian data dengan cepat [ZEI-09].

### 2.2.1.5 *Key*

*Key* yang bernilai unik merupakan salah satu jenis *constraint* yang berfungsi untuk memastikan bahwa suatu objek hanya akan tersimpan satu kali dalam tabel [OKE-09 : 20]. Dengan adanya *key* yang bernilai unik, maka akan membuat setiap baris memiliki identitas yang berbeda antara satu dengan yang lainnya. *Key* seperti ini biasanya disebut dengan *primary key* atau *unique key*. Perbedaan antara *primary key* atau *unique key* adalah :

- Nilai NULL tidak dapat diisikan pada kolom berindex *primary key*, hal ini karena *primary key* merupakan identitas unik yang harus dimiliki setiap kolom dalam tabel. Sedangkan pada kolom berindex *unique key*,

nilai NULL dapat diisikan tetapi hanya satu kali.

- Dalam setiap tabel hanya boleh memiliki satu buah index *primary key* tetapi dapat memiliki banyak index *unique key* [DUB-06 : 8.6].

#### 2.2.1.6 *Foreign Key*

Karena kolom yang memiliki dasar domain yang sama dapat digunakan untuk merelasikan tabel dalam sebuah *database*, maka konsep *foreign key* memungkinkan DBMS untuk menjaga konsistensi di sepanjang baris dari dua tabel atau antar baris dari tabel yang sama.

*Foreign key* merupakan suatu referensi ke sebuah *Key* yang ada pada tabel yang lain, ini berarti baris yang mereferensi akan berisi nilai seperti yang ada dalam baris yang direferensi. *Foreign key* tidak perlu memiliki nilai unik. *Foreign key* secara efektif menggunakan nilai dari tabel yang direferensinya untuk membatasi domain yang digunakan [OKE-09 : 20].

### 2.2.2 *View*

*View* merupakan objek *database* yang didefinisikan dengan menggunakan `SELECT` *statement* yang mengembalikan data yang ingin ditampilkan oleh *view*. *View* biasanya disebut dengan *virtual table* [DUB-06 : 14].

### 2.2.3 *Trigger*

*Trigger* merupakan objek *database* yang diasosiasikan dengan tabel. *Trigger* didefinisikan untuk melakukan suatu *action* ketika terjadi suatu *event* dalam suatu tabel tertentu [DUB-06 : 19].

### 2.2.4 *Routine*

*Routine* merupakan sekumpulan fungsi – fungsi yang disimpan dalam *database*, yang setiap saat dapat dipanggil untuk dieksekusi dalam rangka membantu proses – proses dalam modifikasi *database*. *Routine* dibagi menjadi dua macam yaitu [DUB-06 : 18]:

1. *Stored Procedure*
2. S*tored Function*

### 2.2.5 Sequence

*Sequence* merupakan objek yang berfungsi sebagai pencipta *auto number* dalam *database*. *Sequence* sangat berguna ketika dibutuhkan fungsi untuk membuat angka – angka unik dalam *database* terutama digunakan dalam pengisian data yang bertindak sebagai *primary key* [SMI-03 : 2].

## 2.3 Database Management System (DBMS)

*Database Management System* (DBMS) adalah sekumpulan program komputer yang berfungsi untuk mengontrol penyimpanan, pengelolaan, dan pengambilan data dari suatu *database* [OKE-09 : 2]. DBMS dapat diartikan sebagai perangkat lunak yang dirancang untuk membantu dalam memelihara dan mengelola kumpulan data dalam skala besar. Sistem ini dibuat untuk mengatasi kelemahan sistem pemrosesan yang berbasis berkas. DBMS dirancang untuk dapat melakukan manipulasi data secara lebih mudah. Contoh DBMS adalah Oracle, SQL server 2000/2003, MS Access, MySQL, PostgreSQL dan sebagainya [OKE-09 : 6].

## 2.4 Structured Query Language (SQL)

*Structured Query Language* (SQL) adalah sebuah bahasa yang dipergunakan untuk mengakses data dalam *database* relasional. Bahasa ini secara *de facto* merupakan bahasa standar yang digunakan dalam manajemen *database* relasional. Saat ini, hampir semua server *database* yang ada, mendukung bahasa ini untuk melakukan manajemen datanya [MAT-00 : 78]. Secara umum, SQL terdiri dari dua bahasa, yaitu *Data Definition Language* (DDL) dan *Data Manipulation Language* (DML). Implementasi DDL dan DML berbeda untuk tiap DBMS, namun secara umum implementasi tiap bahasa ini memiliki bentuk standar yang ditetapkan ANSI.

### 2.4.1 *Data Definition Language* (DDL)

DDL digunakan untuk mendefinisikan, mengubah, serta menghapus *database* dan objek-objek yang diperlukan dalam *database*, misalnya tabel, view,

user, dan sebagainya. Secara umum, DDL yang digunakan adalah CREATE untuk membuat objek baru, USE untuk menggunakan objek, ALTER untuk mengubah objek yang sudah ada, dan DROP untuk menghapus objek. DDL biasanya digunakan oleh administrator *database* dalam pembuatan sebuah aplikasi *database* [MAT-00 : 79]. Berikut merupakan *pattern* DDL *statement* CREATE dalam DBMS secara umum :

- **CREATE DATABASE Syntax**

**CREATE DATABASE [IF NOT EXISTS]** db_name
{*create_specification* **[,** *create_specification***]** ...**}**

- **CREATE TABLE Syntax**

**CREATE TABLE [IF NOT EXISTS]** tbl_name
**[(***column_definition*,...**)]**
**[(***index_definition*,...**)]**
**[***table_options***]**

■ ***index_definition*** **(MySQL) :**

**, PRIMARY KEY (***index_col_name*,...**)**
| **, KEY [**index_name**] (***index_col_name*,...**)**
| **, INDEX [**index_name**] (***index_col_name*,...**)**
| **, UNIQUE [INDEX] [**index_name**] (***index_col_name*,...**)**
| **, CONSTRAINT** constraint_name
**FOREIGN KEY (***index_col_name***) [***reference_definition***]**

■ ***index_definition*** **(Oracle) :**

**, CONSTRAITS** [index_name] **PRIMARY KEY (***index_col_name*,...**)**
| **, KEY [**index_name**] (***index_col_name*,...**)**
| **, INDEX [**index_name**] (***index_col_name*,...**)**
| **, UNIQUE [INDEX] [**index_name**] (***index_col_name*,...**)**
| **, CONSTRAINT** constraint_name

**FOREIGN KEY** (index_col_name) [reference_definition]

- ***column_definition*** **:**

col_name *type* **[NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]** [*reference_definition*]

- ***type*** **(MySQL) :**

**TINYINT[(**length**)] [UNSIGNED] [ZEROFILL]**
| **SMALLINT[(**length**)] [UNSIGNED] [ZEROFILL]**
| **MEDIUMINT[(**length**)] [UNSIGNED] [ZEROFILL]**
| **INT[(**length**)] [UNSIGNED] [ZEROFILL]**
| **INTEGER[(**length**)] [UNSIGNED] [ZEROFILL]**
| **BIGINT[(**length**)] [UNSIGNED] [ZEROFILL]**
| **REAL[(**length,decimals**)] [UNSIGNED] [ZEROFILL]**
| **DOUBLE[(**length,decimals**)] [UNSIGNED] [ZEROFILL]**
| **FLOAT[(**length,decimals**)] [UNSIGNED] [ZEROFILL]**
| **DECIMAL(**length,decimals**) [UNSIGNED] [ZEROFILL]**
| **NUMERIC(**length,decimals**) [UNSIGNED] [ZEROFILL]**
| **DATE**
| **TIME**
| **TIMESTAMP**
| **DATETIME**
| **CHAR(**length**) [BINARY | ASCII | UNICODE]**
| **VARCHAR(**length**) [BINARY]**
| **TINYBLOB**
| **BLOB**
| **MEDIUMBLOB**
| **LONGBLOB**
| **TINYTEXT**
| **TEXT**
| **MEDIUMTEXT**
| **LONGTEXT**
| **ENUM(**value1,value2,value3,...**)**
| **SET(**value1,value2,value3,...**)**

- ***type* (Oracle) :**

*character_datatypes :*

```
{ CHAR [ (size) ]
| VARCHAR2 (size)
| NCHAR [ (size) ]
| NVARCHAR2 (size)
}
```

*datetime_datatypes :*

```
{ DATE
| TIMESTAMP [ (fractional_seconds_precision) ]
    [ WITH [ LOCAL ] TIME ZONE ])
| INTERVAL YEAR [ (year_precision) ] TO MONTH
| INTERVAL DAY [ (day_precision) ] TO SECOND
    [ (fractional_seconds_precision) ]
}
```

*large_object_datatypes :*

```
{ BLOB | CLOB | NCLOB | BFILE }
```

*long_and_raw_datatypes :*

```
{ LONG | LONG RAW | RAW (size) }
```

*number_datatypes :*

```
{ NUMBER [ (precision [, scale ]) ]
| FLOAT [ (precision) ]
| BINARY_FLOAT
| BINARY_DOUBLE
}
```

*rowid_datatypes :*

```
{ ROWID | UROWID [ (size) ] }
```

*Numeric type* :

| Name | Storage Size | Description | Range |
|---|---|---|---|
| smallint | 2 bytes | small-range integer | -32768 to +32767 |
| integer | 4 bytes | usual choice for integer | -2147483648 to +2147483647 |
| bigint | 8 bytes | large-range integer | -9223372036854775808 to 9223372036854775807 |

| Name | Storage Size | Description | Range |
|---|---|---|---|
| decimal | variable | user-specified precision, exact | no limit |
| numeric | variable | user-specified precision, exact | no limit |
| real | 4 bytes | variable-precision, inexact | 6 decimal digits precision |
| double precision | 8 bytes | variable-precision, inexact | 15 decimal digits precision |
| serial | 4 bytes | autoincrementing integer | 1 to 2147483647 |
| bigserial | 8 bytes | large autoincrementing integer | 1 to 9223372036854775807 |

**Gambar 2.3** Tabel Tipe Data Numerik PostgreSQL

**Sumber :** The PostgreSQL Development Group. 2005

*Character type* :

| Name | Description |
|---|---|
| character varying(n), varchar(n) | variable-length with limit |
| character(n), char(n) | fixed-length, blank padded |
| text | variable unlimited length |

**Gambar 2.4** Tabel Tipe Data Karakter PostgreSQL

**Sumber :** The PostgreSQL Development Group. 2005

*Date / Time type :*

| Name | Storage Size | Description | Low Value | High Value | Resolution |
|---|---|---|---|---|---|
| timestamp [ (p) ] [ without time zone ] | 8 bytes | both date and time | 4713 BC | 5874897 AD | 1 microsecond / 14 digits |
| timestamp [ (p) ] with time zone | 8 bytes | both date and time, with time zone | 4713 BC | 5874897 AD | 1 microsecond / 14 digits |
| interval [ (p) ] | 12 bytes | time intervals | -178000000 years | 178000000 years | 1 microsecond / 14 digits |
| date | 4 bytes | dates only | 4713 BC | 5874897 AD | 1 day |
| time [ (p) ] [ without time zone ] | 8 bytes | times of day only | 00:00:00 | 24:00:00 | 1 microsecond / 14 digits |
| time [ (p) ] with time zone | 12 bytes | times of day only, with time zone | 00:00:00+1359 | 24:00:00-1359 | 1 microsecond / 14 digits |

**Gambar 2.5** Tabel Tipe Data Tanggal / Waktu PostgreSQL

**Sumber :** The PostgreSQL Development Group. 2005

*Boolean type :*

| Nilai Valid untuk Statement "*true*" | Nilai Valid untuk Statement "*false*" |
|---|---|
| *TRUE* | *FALSE* |
| `true` | `false` |
| `t` | `f` |
| `yes` | `no` |
| `y` | `n` |
| `1` | `0` |

**Gambar 2.6** Tabel Tipe Data *Boolean* PostgreSQL

**Sumber :** The PostgreSQL Development Group. 2005

- ***index_col_name* :**

col_name **[ASC** | **DESC]**

- ***reference_definition*:**

**REFERENCES** tbl_name **[(***index_col_name,...***)]**
**[ON DELETE** *reference_option***]**
**[ON UPDATE** *reference_option***]**

- ***reference_option*:**

**RESTRICT** | **CASCADE** | **SET NULL** | **NO ACTION** | **SET DEFAULT**

- ***table_option* (MySQL) :**

**ENGINE**|**TYPE** = **BDB**|**HEAP**|**ISAM**|**InnoDB**|**MERGE**|**MRG_MYISAM**|**MYISAM**
| **AUTO_INCREMENT =** value
| **[DEFAULT] CHARACTER SET** *charset_name* **[COLLATE** *collation_name***]**

- **CREATE VIEW Syntax**

**CREATE [OR REPLACE] VIEW** view_name **AS** *select_statement*

- **CREATE TRIGGER Syntax**

```
CREATE TRIGGER trigger_name
{ BEFORE | AFTER }
{ INSERT | UPDATE | DELETE }
ON table_name
[FOR EACH ROW]
triggered_statement
```

- **CREATE ROUTINE Syntax**

**Procedure :**

```
CREATE [ OR REPLACE ] PROCEDURE proc_name
[(parameters [{IN | OUT | IN OUT}]datatype [ DEFAULT expr ])]
[characteristics]
{IS | AS}
{program_body};
```

**Function :**

```
CREATE [ OR REPLACE ] PROCEDURE proc_name
[(parameters [{IN | OUT | IN OUT}]datatype [ DEFAULT expr ])]
RETURNS data_type
[characteristics]
{IS | AS}
{ program_body };
```

[MYS : 05]

- **CREATE SEQUENCE Syntax**

```
CREATE SEQUENCE sequence_name
[ { INCREMENT BY | START WITH } number
| { MAXVALUE number | NOMAXVALUE }
| { MINVALUE number | NOMINVALUE }
| { CYCLE | NOCYCLE }
| { CACHE number | NOCACHE }
| { ORDER | NOORDER }];
```

[LOR-07]

### 2.4.2 *Data Manipulation Language* (DML)

DML digunakan untuk memanipulasi data yang ada dalam suatu tabel. Perintah yang umum dilakukan adalah [MAT-00 : 79] :

- SELECT untuk menampilkan data
- INSERT untuk menambahkan data baru
- UPDATE untuk mengubah data yang sudah ada
- DELETE untuk menghapus data

Berikut merupakan *pattern* DML *statement* INSERT dalam DBMS secara umum :

- **INSERT Syntax**

```
INSERT [INTO] tbl_name [(col_name,...)]
VALUES ({expr | DEFAULT},...),(...),...
```

Atau:

```
INSERT [INTO] tbl_name SET col_name ={expr | DEFAULT}, ...
```

[MYS : 05]

## 2.5 Java

Bahasa pemrograman Java merupakan salah satu bahasa pemrograman yang telah diakui kehandalannya. Selain itu, Java menambahkan fitur – fitur lain seperti *garbage collection* sebagai pengatur memori otomatis, *multithreading*, dan kemampuan dalam keamanan. Java secara umum terdiri dari tiga buah komponen utama yaitu :

1. Bahasa pemrograman Java
2. Java *library* yang terdiri dari *classes* dan *interfaces*
3. *Java Virtual Machine* (JVM)

Salah satu yang menjadi keuntungan terbesar yang ditawarkan dari teknologi Java adalah sifatnya yang *portable.* Sebuah aplikasi yang ditulis dengan bahasa pemrograman Java akan dapat berjalan pada sebagian besar *platform* yang ada. Seorang *programmer* tidak lagi harus menulis satu program untuk dijalankan dalam sistem operasi Macintosh, dan program lain untuk dijalankan dalam

Windows *machine*, dan program yang lain lagi untuk sistem operasi Solaris atau Linux. Dengan kata lain, dengan teknologi Java, seorang *developer* cukup menulis satu program untuk dijalankan pada *platform* manapun.

JVM yang menjadikan Java dapat menghasilkan sebuah program yang *multi-platform.* Daripada secara langsung di-*compile* menjadi bahasa mesin yang hasilnya akan berbeda berdasarkan sistem operasi dan rancangan komputer yang digunakan, Java *code* di-*compile* kedalam bentuk yang disebut *bytecode* yang membuat hasilnya lebih *portable.* Dengan bahasa pemrograman lain, sebuah program di-*compile* menjadi bahasa mesin yang dimengerti oleh komputer. Permasalahannya adalah ketika koputer dengan pengaturan instruksi mesin yang berbeda membaca program hasil kompilasi tersebut, maka program itu tidak akan dapat berjalan. Disisi lain, Java *code* yang di-*compile* menjadi *bytecode*, dengan JVM yang akan melakukan translasi *bytecode* menjadi bahasa yang dimengerti komputer, maka program tersebut akan dapat berjalan di *platform* mana saja [FIS-03 : 1.6].

## 2.6 JDBC

### 2.6.1 Pengertian JDBC

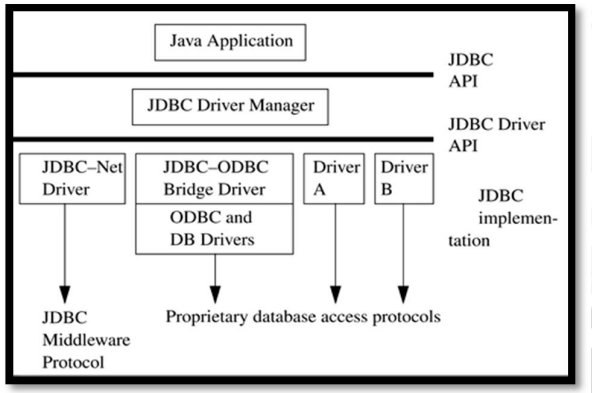
JDBC merupakan sebuah nama merek dagang atau *trademark* dari Sun Microsystem. Meskipun demikian, banyak orang menganggap JDBC sebagai sebuah singkatan dari "*Java Database Connectivity*" yang merupakan bagian dari teknologi Java dan digunakan untuk pengolahan *database*. Pengertian JDBC sendiri secara umum adalah sebuah antarmuka antara *database* relasional dengan Java. Dalam dunia Java *programming*, JDBC merupakan sebuah *Application Programming Interface* (API) yang digunakan untuk kebutuhan yang berhubungan dengan *Database Management System* (DBMS) relasional (seperti Oracle, MySQL, PostgreSQL, Sybase, dan DB2), baik itu kebutuhan untuk koneksi maupun kebutuhan untuk pengolahan *database*. Pemrograman JDBC dapat dijelaskan dalam beberapa langkah sederhana berikut [FIS-03 : 1.4] :

1. Mengimpor paket yang dibutuhkan

2. Pendaftaran JDBC *driver*
3. Membuka koneksi ke *database*
4. Membuat *Statement* / *PreparedStatement* / *CallableStatement* objek
5. Mengeksekusi sebuah *query* SQL dan memberikan kembalian berupa objek *ResultSet*
6. Pengolahan objek *ResultSet*
7. Menutup *Statement* dan *ResultSet* objek
8. Menutup koneksi

### 2.6.2 JDBC Driver

Secara garis besar, JDBC terdiri dari dua buah antarmuka utama yaitu JDBC API untuk hubungan dengan penulisan aplikasi dan JDBC *Driver* API *lower-level* untuk hubungan dengan *driver.* JDBC *Driver* API ini yang diimplementasikan untuk mengatur koneksi dengan *data source / database* yang sedang terjadi. Gambar 2.1 menunjukkan rancangan dari JDBC. *DriverManager class* digunakan untuk membuka koneksi ke *database* melalui JDBC *Driver* yang sudah didaftarkan dalam *DriverManager* sebelum koneksi dapat dilakukan. Ketika koneksi sedang dilakukan, *DriverManager* memilih *driver* dari daftar yang tersedia untuk disesuaikan dengan koneksi *database* yang diminta. Setelah koneksi terbentuk, proses *query* dan pengambilan hasil dilakukan secara langsung dengan JDBC *Driver*. JDBC *Driver* harus mengimplementasikan kelas – kelas JDBC sehingga memiliki fungsi untuk melakukan proses – proses sesuai dengan *database* tertentu, dimana JDBC yang sudah dibangun dengan spesifikasi tertentu tersebut menjamin bahwa *driver*-nya akan dapat berjalan sesuai dengan yang diharapkan. Singkatnya, JDBC adalah sebuah API *database*-independen untuk mengakses *database* relasional. Sebuah *syntax* SQL di-*passing* melalui Java *method* dalam kelas JDBC (*package* java.sql dan javax.sql), kemudian didapatkan objek kembalian (seperti ResultSet, DatabaseMetaData, dan ResultSetMetaData) yang merupakan hasil *query* [FIS-03 : 1.4].




**Gambar 2.7** Implementasi JDBC *Driver*

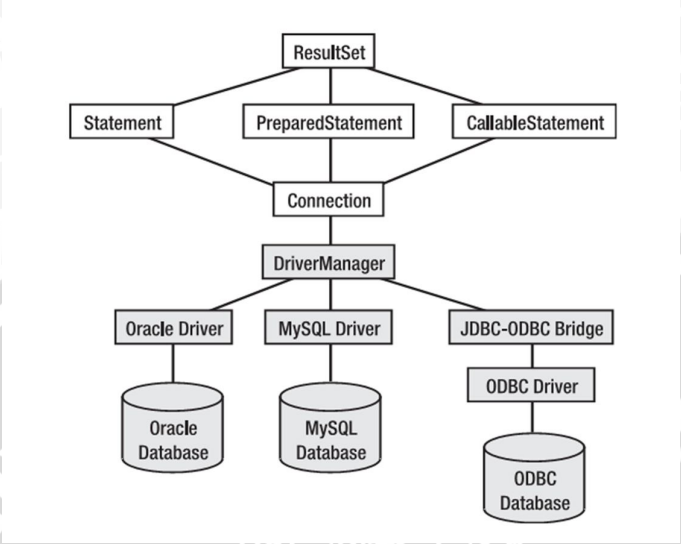
**Sumber :** Fisher, Maydene, Jon Ellis, Jonathan Bruce. 2003

### 2.6.3 Komponen Utama JDBC

JDBC API memiliki beberapa komponen utama yang berhubungan dengan mekanisme pengolahan data baik itu untuk berkomunikasi dengan server *database*, melakukan koneksi, mengirim perintah SQL, mendapatkan data, menutup koneksi, serta penanganan *error* yang mungkin terjadi dalam pengolahan data. Komponen utama JDBC tersebut antara lain [FIS-03 : 1.4] :

1. *Driver* adalah komponen untuk menangani komunikasi dengan *database* server
2. *DriverManager* adalah komponen untuk menangani objek *Driver* dimana objek *DriverManager* juga mengabstraksi detail dari proses kerja objek *Driver*
3. *Connection* adalah komponen untuk merepresentasikan koneksi secara fisik ke *database*
4. *Statement* adalah komponen untuk mengirim perintah-perintah SQL ke *database*
5. *ResultSet* adalah komponen untuk menyimpan data yang didapat dari *database* setelah perintah SQL dijalankan dengan menggunakan komponen *Statement*
6. *SQLException* adalah komponen untuk menangangi kesalahan-kesalahan (*error*) yang mungkin terjadi dalam pengolahan *database*

Gambar 2.4 menunjukkan hubungan *interfaces* dan *classes* dalam JDBC [PAR-06 : 14] :




**Gambar 2.8** *Interfaces* dan *Classes* pada JDBC

**Sumber :** Parsian, Mahmoud. 2006

## 2.7 Metadata

Data merupakan jantung dari sebuah DBMS. Ada dua macam data, yang pertama adalah kumpulan informasi yang dibutuhkan oleh organisasi dan yang kedua adalah *metadata* atau kumpulan informasi tentang *database*. Metadata adalah informasi terstruktur yang menggambarkan, menjelaskan, menempatkan, atau membuat lebih mudah untuk mengambil, menggunakan, dan mengelola sumber informasi. Metadata sering disebut "data tentang data" atau "informasi tentang informasi" [NAT-04: 1]. Metadata *database* biasanya disimpan dalam *data dictionary* atau *information_schema* [MAT-00 : 4].

Tujuan dari metadata dalam *database* adalah untuk membuat *database* objek lebih mudah diakses oleh pengguna dan untuk memberikan informasi dasar tentang hal – hal yang ada dalam koleksi *database* tersebut. Metadata dalam *database* biasanya digunakan oleh *tools* atau *developers* yang menulis program tentang *database* profesional. Sebagai contoh informasi yang ingin didapat melalui metadata, digambarkan dalam beberapa pertanyaan berikut ini :

1. Apa saja nama tabel dan *view* yang ada dalam *database*?

2. Apa saja nama dan tipe dari kolom yang ada dalam tiap tabel dan *view*?
3. Apa saja *stored procedure* yang ada dalam *database*?

### 2.7.1 MySQL Metadata

MySQL metadata merupakan informasi tentang data yang ada pada DBMS MySQL. Metadata pada MySQL ini dapat diakses melalui *schema* bernama INFORMATION_SCHEMA. Hingga sekarang, MySQL telah menyediakan akses metadata melalui sekumpulan perintah-perintah SHOW [PEL-05 : 3].

### 2.7.2 Oracle Metadata

Oracle metadata adalah informasi yang terkandung dalam DBMS Oracle, dimana informasi tersebut menyangkut seluruh objek yang ada di dalam *database* Oracle. Informasi ini dapat digunakan untuk menemukan seluruh tabel yang ada, menemukan seluruh daftar prosedur yang tersimpan, dan informasi – informasi lain menyangkut berbagai macam objek dalam *database* Oracle. Oracle menyimpan metadata pada tabel *data dictionary* (dapat diakses melalui *built-in views*).

### 2.7.3 PostgreSQL Metadata

PostgreSQL metadata merupakan informasi tentang objek – objek yang ada dalam DBMS PostgreSQL. Tidak berbeda dari MySQL, informasi – infromasi yang berhubungan dengan metadata dari PostgreSQL disimpan pada *schema* bernama INFORMATION_SCHEMA.

## 2.8 Application Programming Interface (API)

*Application Programming Interface* (API) merupakan kumpulan *rotine*, protokol, dan *tools* yang dapat digunakan dalam pembangunan aplikasi. Aplikasi dapat dibangun menggunakan fungsi dan prosedur yang terdapat pada API untuk memudahkan pelaksanaan aksi tertentu. API bersifat *reusable* dan *independent*

terhadap aplikasi. API yang bagus memiliki beberapa karakteristik [BLO : 05] :

1. Mudah dipelajari
2. Mudah digunakan
3. Tidak mudah disalahgunakan
4. Kodenya mudah dibaca dan dipelihara
5. Cukup *powerfull* untuk memenuhi *requirement*
6. Mudah di-*extends*

Langkah pertama dalam merancang API adalah menentukan *requirement*. Gambaran untuk *requirement* bisa didapatkan dari *use-case* dan sumber lain seperti masukan dari orang yang ditargetkan akan menggunakan API yang dikembangkan. Pada awalnya, spesifikasi yang ditentukan untuk API sedikit tetapi *general*, sehingga akan lebih mudah untuk dikembangkan. Langkah selanjutnya adalah desain, analisis, dan implementasi. Proses tersebut dan *requirement* dapat dilakukan secara paralel. Dari spesifikasi API yang dibuat berdasarkan *requirement*, mulai dikembangkan API secara sederhana. Dari proses pengembangan API ini, diidentifikasi kebutuhan tambahan yang dapat dimasukkan dalam spesifkasi dan juga spesifikasi yang ternyata tidak dibutuhkan dalam pengembangan [BLO : 05].

## 2.9 Reverse Engineering Database

*Reverse engineering* merupakan suatu langkah dalam siklus hidup sistem infromasi. *Reverse engineering database* biasanya dimaksudkan untuk pendokumentasian ulang, restrukturisasi, pemeliharaan, atau perluasan aplikasi warisan. Berikut merupakan beberapa tujuan utama dari *reverse engineering database* [HAI-02 : 2] :

- Pengetahuan akuisisi dalam pengembangan sistem
- Pemeliharaan sistem
- Rekayasa ulang sistem
- Perluasan sistem
- Migrasi sistem
- Integrasi sistem

- Kualitas penilaian
- Ekstraksi data / konversi
- Administrasi data
- Penggunaan kembali komponen

# BAB III
# METODOLOGI

Pada bab ini dijelaskan langkah-langkah yang akan dilakukan dalam perancangan, implementasi dan pengujian dari aplikasi perangkat lunak yang akan dibuat. Kesimpulan dan saran disertakan sebagai catatan atas aplikasi dan kemungkinan arah pengembangan aplikasi selanjutnya.

## 3.1 Perancangan dan Implementasi Sistem

Perancangan API dilakukan dengan mengidentifikasi *interfaces* dan *classes* yang dibutuhkan dan dimodelkan dalam *class diagram*. Implementasi API dilakukan dengan mengacu kepada perancangan serta dilakukan dengan menggunakan bahasa pemrograman berorientasi objek, yaitu menggunakan bahasa pemrograman Java (Java *Standard Edition*) dan JDBC. Perancangan dan implementasi API selanjutnya dijelaskan dalam diagram alur proses yang menjelaskan tentang hubungan interaksi antar elemen (objek) yang telah diidentifikasi.

Perancangan dan implementasi untuk sistem API *reverse engineering database* ini secara garis besar meliputi :

- Proses mendapatkan *connection* dengan DBMS
- Proses pemeriksaan DBMS vendor
- Proses mendapatkan metadata
- Proses melakukan *sorting* tabel – tabel relasi dalam *database*
- Proses *generate* SQL *script*
- Proses pembuatan *output* berisi SQL *script*

API *reverse engineering database* ini secara umum dapat digambarkan dalam sebuah diagram blok sistem *reverse engineering database* seperti pada gambar di bawah ini :


**Gambar 3.1** Diagram Blok Sistem *Reverse Engineering Database*

## 3.2 Pengujian dan Analisis

Pengujian API pada skripsi ini dilakukan agar dapat menunjukkan bahwa API telah mampu bekerja sesuai dengan spesifikasi dari kebutuhan yang melandasinya. Pengujian yang dilakukan antar lain :

1. Pengujian bahwa API dapat melakukan *reverse engineering database* dengan menggunakan koneksi yang didapat dari multi DBMS (MySQL, Oracle, PostgreSQL) menjadi SQL *script* dengan struktur *database* yang benar.
2. Pengujian bahwa SQL *script* hasil *reverse engineering database* dari ToREn API dapat dieksekusi dengan menghasilkan struktur yang sama pada DBMS asalnya.
3. Pengujian ToREn API terhadap masukan koneksi DBMS yang salah.
4. Kesimpulan hasil pengujian.

## 3.3 Pengambilan Kesimpulan

Pengambilan kesimpulan dilakukan setelah semua tahapan perancangan, implementasi dan pengujian sistem telah selesai dilakukan. Kesimpulan diambil dari hasil pengujian dan analisis terhadap sistem yang dibangun. Tahap terakhir dari penulisan adalah saran yang dimaksudkan untuk memperbaiki kesalahan-kesalahan yang terjadi dan menyempurnakan penulisan serta untuk memberikan pertimbangan atas pengembangan sistem selanjutnya.

# BAB IV
# PERANCANGAN DAN IMPLEMENTASI

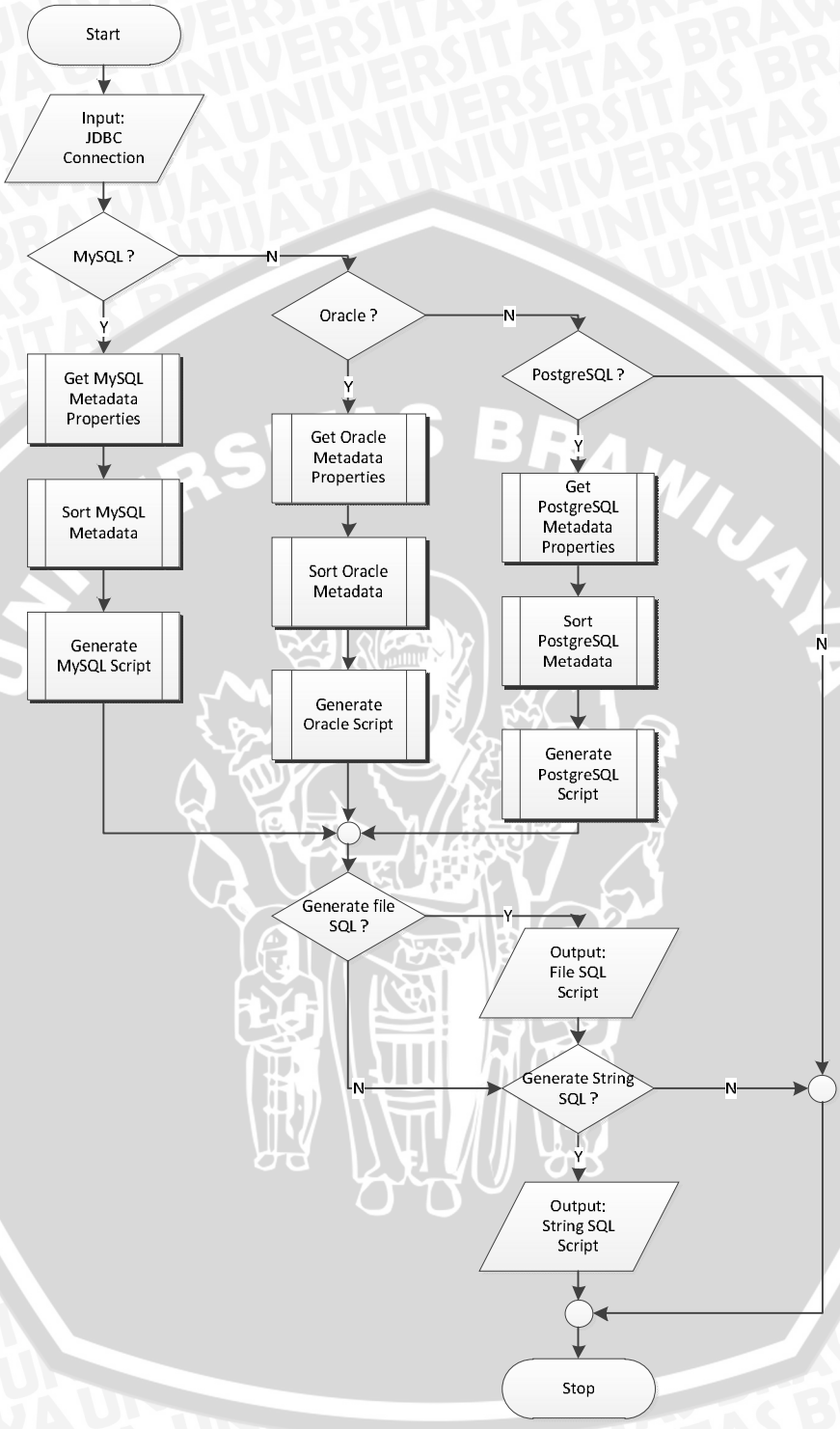
Perancangan dan implementasi *Application Programming Interface (*API*)* untuk Melakukan *Reverse Engineering Database* dari *Multi Database Management System (*DBMS*)* Menjadi *Sturctured Query Language* (SQL*) Script* dengan Java ini dikerjakan dengan beberapa tahap. Tahapan tersebut meliputi proses mendapatkan koneksi dari DBMS, proses pemeriksaan DBMS vendor, proses mendapatkan metadata, proses melakukan *sorting database metadata component*, proses *generate* SQL *script*, dan proses pembuatan *output* berisi SQL *script*. Pembuatan secara bertahap tersebut bertujuan untuk memudahkan penganalisaan API pada setiap bagian maupun secara keseluruhan. Untuk selanjutnya, *Application Programming Interface (*API*)* untuk Melakukan *Reverse Engineering Database* dari *Multi Database Management System (*DBMS*)* Menjadi *Sturctured Query Language* (SQL*) Script* dengan Java ini diberi nama ToREn API.

## 4.1 Perancangan Secara Umum

Perancangan secara global merupakan tahap awal sebagai acuan dalam perancangan ToREn API yang akan dibuat. Perancangan ini diawali dengan pendefinisian alur kerja ToREn API dengan menggunakan penggambaran diagram alir umum sistem dan keterangan cara kerja sistem. Pada perancangan desain, dilakukan identifikasi terhadap kelas-kelas yang dibutuhkan, yang dimodelkan dalam *class diagram*.

### 4.1.1 Diagram Alir Umum Sistem

Sistem ToREn API ini terdiri dari beberapa komponen yang dapat digambarkan secara umum dengan model seperti pada Gambar 4.1 :




**Gambar 4.1** Diagram Alir Umum Sistem

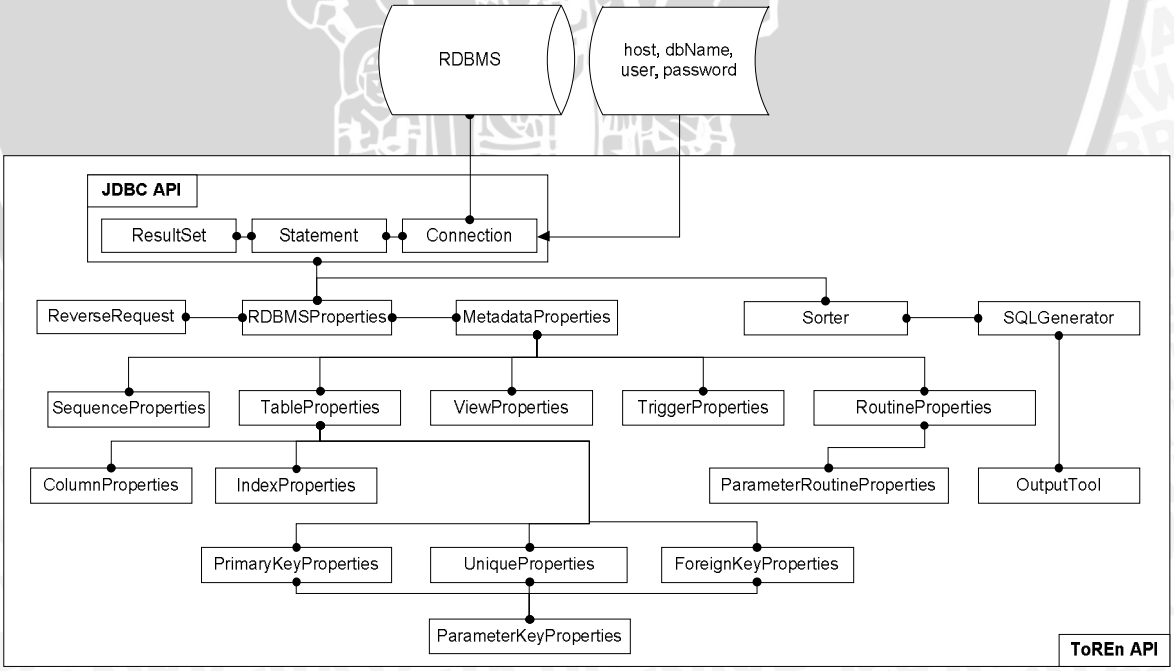
Keterangan:

1. Mendapatkan koneksi JDBC sebagai jalur komunikasi antara ToREn API dan DBMS.

2. Menentukan vendor DBMS dari koneksi yang didapat sebagai parameter acuan untuk menentukan alur program selanjutnya.
3. Mendapatkan metadata yang meliputi informasi detil tabel beserta relasinya, informasi view, informasi trigger, dan informasi *stored procedure* dalam *database*.
4. Melakukan *sorting database metadata components* untuk menyusun urutan komponen – komponen *database* yang tepat.
5. *Generate* SQL *script* yang menghasilkan susunan *script* SQL yang siap digunakan.
6. Pembuatan *output* berisi SQL *script* berupa String atau berupa *file sql*.

### 4.1.2 Desain Umum Sistem

Garis besar perancangan sistem dari ToREn API terdiri dari beberapa langkah pembuatan *interface* dan *class*. Perancangan desain umum sistem ToREn API ini dapat digambarkan seperti ditunjukkan dalam Gambar 4.2 :




**Gambar 4.2** Desain Umum Sistem

### 4.1.3 Cara Kerja

Alur kerja ToREn API berawal dari sebuah koneksi dengan DBMS yang dibuat dari JDBC API. Dengan objek Connecton inilah ToREn API dapat mengakses *database*. Dari koneksi yang didapat, dilakukan pemeriksaan vendor DBMS. Informasi yang didapat dari proses ini berguna sebagai acuan untuk menentukan proses selanjutnya karena API ini memiliki kemampuan untuk melakukan *reverse engineering* dari 3 buah DBMS yaitu MySQL, Oracle, dan PostgreSQL.

Gambar 4.2 menunjukkan struktur kelas yang ada dalam ToREn API beserta hubungan ToREn API sendiri dengan RDBMS. JDBC API merupakan paket API yang disertakan dalam ToREn API. Selain objek Connecton, terdapat objek Statement dan objek ResultSet yang digunakan ToREn API. Kedua objek tersebut digunakan dalam melakukan *querying* informasi metadata pada RDBMS. Proses *querying* dilakukan dengan menggunakan objek Connecton sebagai jalur koneksi antara ToREn API dan RDBMS, menggunakan objek Statement untuk mengirim perintah *query*, dan menggunakan objek ResultSet sebagai penampung hasil *query* dari RDBMS. Gambaran bentuk dari objek ResultSet adalah seperti layaknya suatu tabel yang terdiri dari baris dan kolom. Selain tabel, objek ResultSet memiliki sebuah *pointer* yang siap diarahkan pada *tuple* di dalam tabel, sehinga melalui objek ResultSet didapatkan informasi metadata yang dibutuhkan.

Informasi metadata yang dibutuhkan dalam proses *reverse engineering database* menggunakan ToREn API meliputi informasi detil tabel beserta relasinya, Informasi-informasi tersebut didapatkan dari *information_schema* atau *data_dictionary* yang ada di masing–masing DBMS, dimana di dalamnya menyimpan seluruh informasi tentang *database* dalam DBMS tersebut. Informasi–informasi tersebut didapat dengan cara mengirimkan *query* untuk dieksekusi yang selanjutnya data hasil *query* ditampung dalam objek–objek terstruktur yang telah disediakan.

Objek–objek terstruktur yang disediakan oleh ToREn API berawal dari objek kelas RDBMSProperties yang berfungsi sebagai penampung objek Connection. Dari objek RDBMSProperties, selanjutnya diciptakan objek MetaDataProperties sebagai objek yang mengawali proses *querying* untuk

mendapatkan informasi metadata berupa informasi tabel – tabel yang selanjutnya disimpan pada objek TableProperties, informasi seluruh *view* yang selanjutnya disimpan pada objek ViewProperties, informasi seluruh *trigger* yang selanjutnya disimpan pada objek TriggerProperties, informasi seluruh *routine* beserta parameter yang dibutuhkan yang selanjutnya disimpan pada objek RoutineProperties dan ParameterRoutineProperties, serta informasi seluruh *sequrnce* yang selanjutnya disimpan pada objek SequenceProperties. Kumpulan setiap objek *properties* selanjutnya disimpan dalam objek List yang disediakan di dalam objek MetaDataProperties.

Objek TableProperties, selain berfungsi sebagai penampung informasi metadata tentang tabel, objek ini juga melakukan tugas untuk mengambil seluruh informasi metadata tentang objek penyusun tabel yang terdiri dari kolom, *index* dan *keys* beserta parameter yang dibutuhkannya. Informasi objek penyusun tabel ini disimpan pada *properties* yang telah disediakan dan dikumpulkan dalam objek List yang disediakan masing–masing objek TableProperties.

Dari data yang telah tertampung dalam objek, dapat dilakukan proses *sorting* (pengurutan) komponen–komponen metadata dari *database*. Proses *sorting* dilakukan pada objek RelationalTableSorter. Proses *sorting* ini menghasilkan urutan objek-objek tabel dari tabel yang tidak memiliki referensi atau tabel master yang direferensi, hingga objek-objek tabel yang membutuhkan referensi. Proses *sorting* bertujuan untuk merencanakan penciptaan urutan tabel–tabel, terutama tabel yang memiliki relasi dengan tabel lain, sedemikian sehingga tersusun secara berurutan dan siap di-*generate* menjadi SQL *script*. Proses *generate* SQL *script* yang dilakukan oleh objek SQLGenerator terbagi menjadi tiga macam cara berdasarkan vendor DBMS yang digunakan. *Generate* SQL *script* menghasilkan susunan *script* SQL yang siap digunakan.

Proses terakhir adalah proses yang dilakukan olek objek OutputCreator untuk menghasilkan *output* SQL *script* yang dapat berupa *file sql* atau berupa String yang siap dikeluarkan melalui *tool output. Output* yang dihasilkan dapat berupa keseluruhan struktur dan data, atau dapat pula berupa pilihan beberapa objek saja. Untuk memilih beberapa objek yang akan di-*reverse engineering*,

maka disediakan objek ReverseRequest yang berfungsi sebagai penampung nama–nama objek yang dipilih.

## 4.2 Perancangan dan Implementasi Perangkat Lunak

Perancangan perangkat lunak dibangun dengan menggunakan bahasa pemrograman Java. Karena ToREn API merupakan kumpulan *interface* dan *class*, maka pada perancangan ToREn API ini dijelaskan tentang *interfaces* dan *classes* yang ada beserta atribut, prosedur / fungsi, serta alur kerja-nya.

Pembahasan prosedur / fungsi secara terperinci hanya disertakan pada prosedur / fungsi utama saja. Akan terdapat fungsi getter() sebagai perwakilan dari seluruh fungsi, yang gunanya adalah sebagai media pengambil atribut dalam kelas. Terdapat pula prosedur setter() sebagai perwakilan dari seluruh prosedur, yang gunanya adalah sebagai media pemberi nilai atribut dalam kelas.

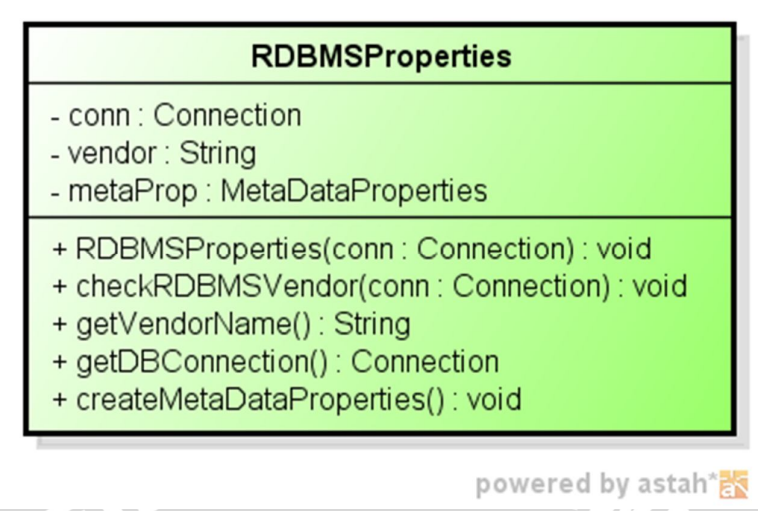
### 4.2.1 RDBMSProperties

RDBMSProperties merupakan kelas yang menerima dan menyimpan objek koneksi JDBC. Setelah koneksi didapatkan, objek RDBMSProperties melakukan pemeriksaan nama vendor dengan cara mengambil informasi String dari objek koneksi JDBC. Setelah nama vendor didapat, nama vendor disimpan dalam atribut objek ini. Nama vendor disimpan karena akan berguna dalam pemeriksaan koneksi pada proses–proses lain.

Proses selanjutnya adalah pembuatan objek MetaDataProperties sesuai dengan vendor tertentu. Hal ini karena setiap vendor memiliki ciri khusus dalam pengambilan informasi metadata.

RDBMSProperties juga merupakan objek tempat pemanggilan proses pengurutan tabel relasional. Setelah objek MetaDataProperties mendapatkan seluruh informasi metadata, maka RDBMSProperties akan membuat objek RelationalTableSorter dan memakai objek ini untuk melakukan pengurutan tabel-tabel yang berelasi.

Gambar 4.3 menunjukkan *class diagram* RDBMSProperties :



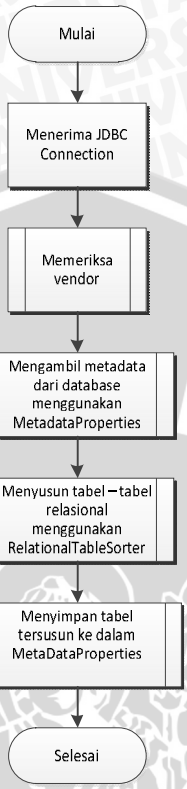

**Gambar 4.3** RDBMSProperties *Class Diagram*

Informasi metadata yang disimpan dalam atribut kelas ini antara lain :

- **conn** : menyimpan objek koneksi *database*.
- **vendor** : menyimpan nama vendor dari koneksi yang diperoleh.
- **metaProp** : menyimpan objek MetaDataProperties yang menyimpan seluruh informasi metadata dari *database*.

Gambar 4.4 dan 4.5 menunjukkan alur kerja (*flow chart*) prosedur / fungsi utama dalam kelas RDBMSProperties :

✓ *Consturctor* **RDBMSProperties(Connection conn)**

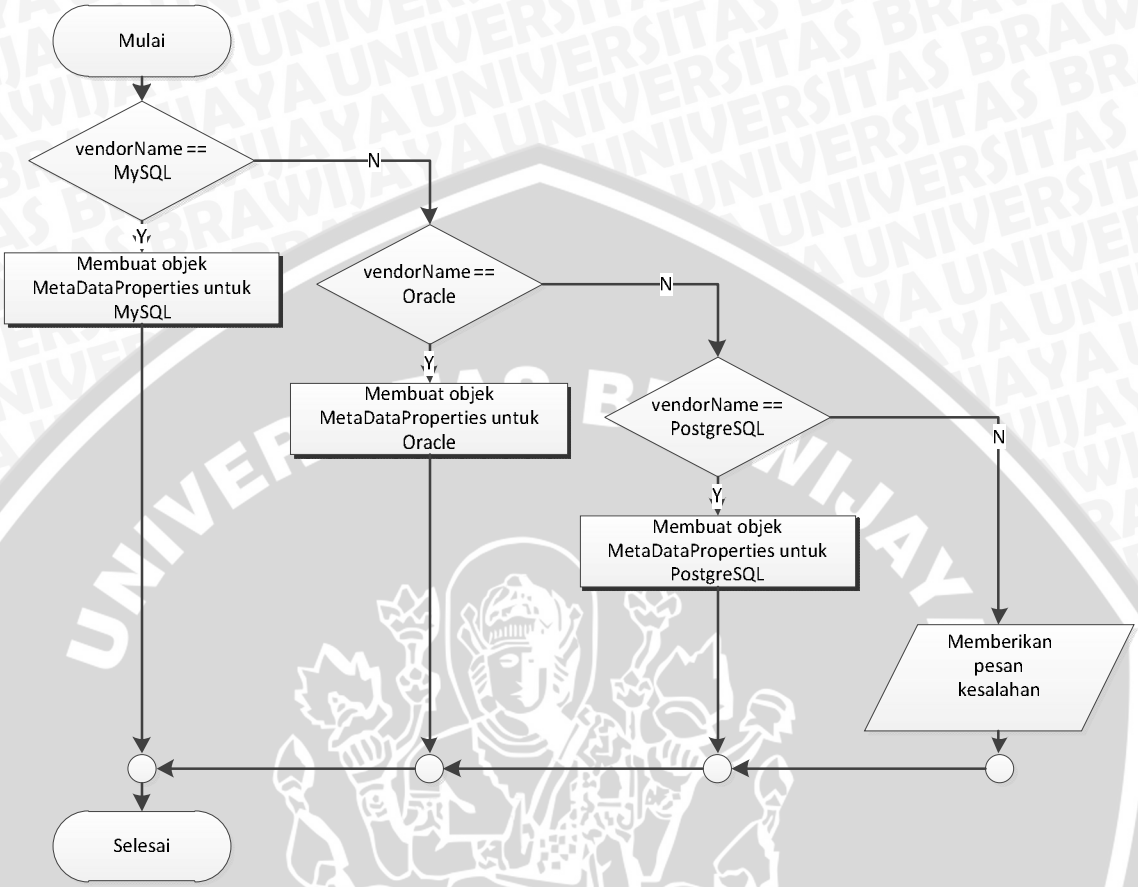



**Gambar 4.4** *Flow Chart Constructor* RDBMSProperties

Penjelasan diagram alir RDBMSProperties adalah sebagai berikut :

1. Menerima koneksi DBMS dalam bentuk objek Connection JDBC.
2. Memeriksa vendor dari koneksi yang diterima dengan melakukan *split* String dari objek koneksi.
3. Mengambil metadata dari *database* dengan menggunakan kelas MetadataProperties.
4. Informasi metadata tabel relasional yang sudah disimpan dalam MetadataProperties, disusun menggunakan kelas RelationalTableSorter.
5. Hasil susunan tabel relasi disimpan kembali pada MetadataProperties untuk selanjutnya akan di-*generate* menjadi SQL *script.*

✓ *Method* **createMetaDataProperties()**




**Gambar 4.5** *Flow Chart* Prosedur createMetaDataProperties

Penjelasan diagram alir createMetaDataProperties adalah sebagai berikut :

1. Menyocokkan nama vendor yang didapat dari proses *splitting* string Connection dengan tiga nama DBMS yang digunakan.
2. Apabila hasil penyocokan sesuai dengan salah satu nama vendor, maka proses pengambilan metadata dapat dilakukan, apalbila tidak maka disediakan kelas *exception handling* untuk menampilkan pesan kesalahan.

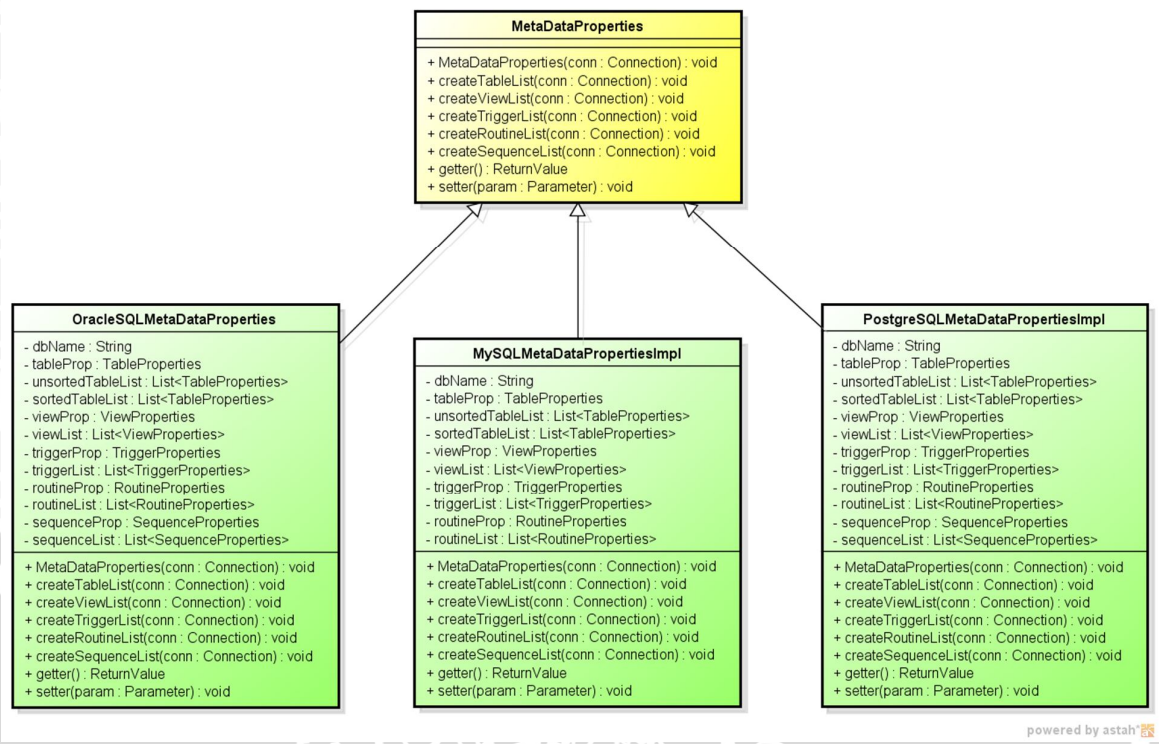
#### 4.2.1.1 MetaDataProperties

MetaDataProperties merupakan *interface* yang selanjutnya di implementasikan berdasarkan vendor tertentu menjadi kelas yang berfungsi untuk melakukan proses pengambilan informasi metadata dari koneksi *database* yang diterima. Alur proses kelas MetaDataProperties berawal dari pengambilan nama *database* yang terkoneksi. Selanjutnya MetaDataProperties mengambil informasi metadata *database*. Informasi metadata yang diambil, dikelompokkan oleh MetaDataProperties sesuai dengan jenisnya yaitu :

- SequenceProperties
- TableProperties
- ViewProperties
- TriggerProperties
- RoutineProperties

Untuk vendor MySQL, tidak ada pengambilan informasi metadata SequenceProperties karena dalam vendor ini tidak menggunakan fungsi *sequence* untuk melakukan *auto numbering*, melainkan menggunakan *Auto Increment* yang langsung menjadi atribut dari objek kolom.

Struktur penyimpanan informasi metadata setiap vendor berbeda –beda, maka dari itu, MetaDataProperties untuk setiap vendor dalam ToREn API ini memiliki implementasi yang berbeda pula. Sehingga, perancangan MetaDataProperties diawali dengan pembuatan *interface* yang selanjutnya diimplementasikan kedalam tiga buah kelas MetaDataProperties sesuai vendornya. Gambar 4.6 menunjukkan *class diagram* MetaDataProperties :




**Gambar 4.6** MetaDataProperties *Class Diagram*

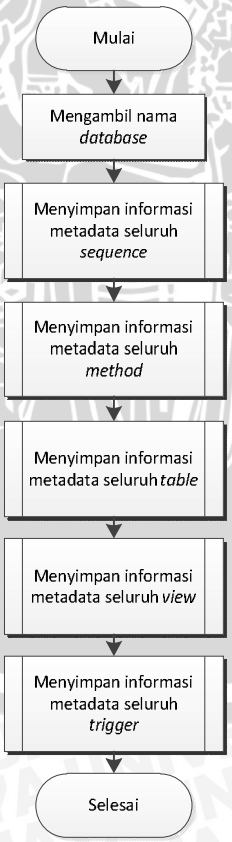
Informasi metadata yang disimpan dalam atribut kelas ini antara lain :

- **dbName** : menyimpan nama *database*.
- **tableProp** : menyimpan objek TableProperties yang berisi informasi metadata dari suatu tabel.
- **sortedTableProp** : menyimpan list seluruh objek TableProperties yang belum tersusun berdasarkan relasinya.
- **sortedTableList** : menyimpan list seluruh objek TableProperties yang telah tersusun berdasarkan relasinya.
- **viewProp** : menyimpan objek ViewProperties yang berisi informasi metadata dari suatu *view.*
- **viewList** : menyimpan *list* seluruh objek ViewProperties dalam *database.*
- **triggerProp** : menyimpan objek TriggerProperties yang berisi informasi metadata dari suatu *trigger*.

- **triggerList** : menyimpan list seluruh objek TriggerProperties dalam *database.*
- **routineProp** : menyimpan objek RoutineProperties yang berisi informasi metadata dari suatu metode.
- **routineList** : menyimpan list seluruh objek RoutineProperties dalam *database.*
- **sequenceProp** : menyimpan objek SequenceProperties yang berisi informasi metadata dari suatu *sequence*.
- **sequenceList** : menyimpan list seluruh objek SequenceProperties dalam *database.*

Gambar 4.7 - 4.8 menunjukkan alur kerja (*flow chart*) prosedur / fungsi utama dalam kelas MetaDataProperties :

✓ *Consturctor* **MetaDataProperties(Connection conn)**

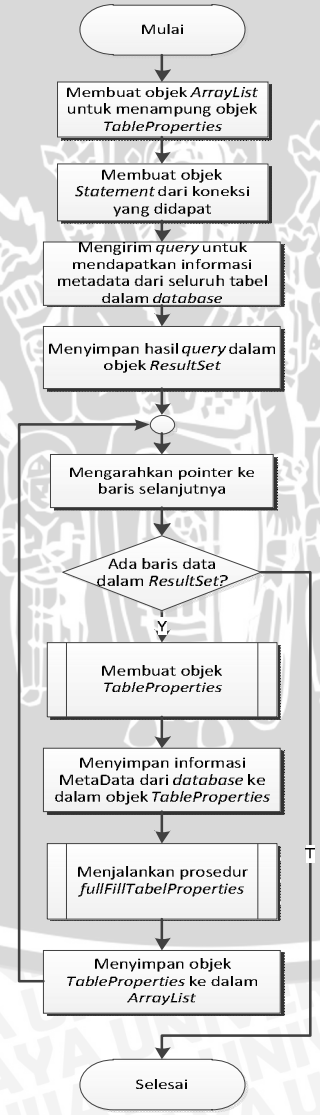



**Gambar 4.7** *Flow Chart Constructor* MetaDataProperties

Penjelasan diagram alir *constructor* MetaDataProperties adalah sebagai berikut :

1. Mengambil nama *database* dari pengaturan koneksi yang dibuat.
2. Menjalankan subfungsi untuk pengambilan informasi metadata *sequences.*
3. Menjalankan subfungsi untuk pengambilan informasi metadata *methods.*
4. Menjalankan subfungsi untuk pengambilan informasi metadata *tables.*
5. Menjalankan subfungsi untuk pengambilan informasi metadata *views.*
6. Menjalankan subfungsi untuk pengambilan informasi metadata *triggers.*

✓ *Method* **createTableList(Connection conn)**




**Gambar 4.8** *Flow Chart* Prosedur createTabelList

Penjelasan diagram alir createTabelList adalah sebagai berikut :

1. Membuat objek bertipe array untuk menampung objek-objek TableProperties.
2. Membuat objek Statement dari koneksi yang didapat, untuk digunakan dalam pengiriman *query* dari API menuju DBMS tertentu.
3. Mengirim *query* untuk mengambil metadata dari suatu tabel.
4. Hasil *query* berupa objek ResultSet disimpan satu per satu kedalam objek TableProperties.
5. Seluruh objek TableProperties disimpan ke dalam objek array.

Untuk mendapatkan informasi metadata tabel dalam prosedur createTabelList, maka dilakukan pengiriman *query* pada DBMS tujuan. *Query* yang dikirimkan pada masing-masing DBMS untuk mendapatkan informasi tabel yaitu :

**a. MySQL**

*Query* berikut berfungsi untuk mengambil informasi metadata *table name, engine name,* dan *auto increment* dari *information_schema.TABLES* :

```
"SELECT t.TABLE_NAME, t.TABLE_TYPE, t.ENGINE, t.AUTO_INCREMENT FROM
information_schema.TABLES t, information_schema.
COLLATION_CHARACTER_SET_APPLICABILITY c WHERE t.TABLE_SCHEMA =
'"+dbName+"' AND t.TABLE_COLLATION = c.COLLATION_NAME"
```

**b. Oracle**

*Query* berikut berfungsi untuk mengambil informasi metadata *table name* dari *user_tables* :

```
"SELECT table_name FROM user_tables"
```

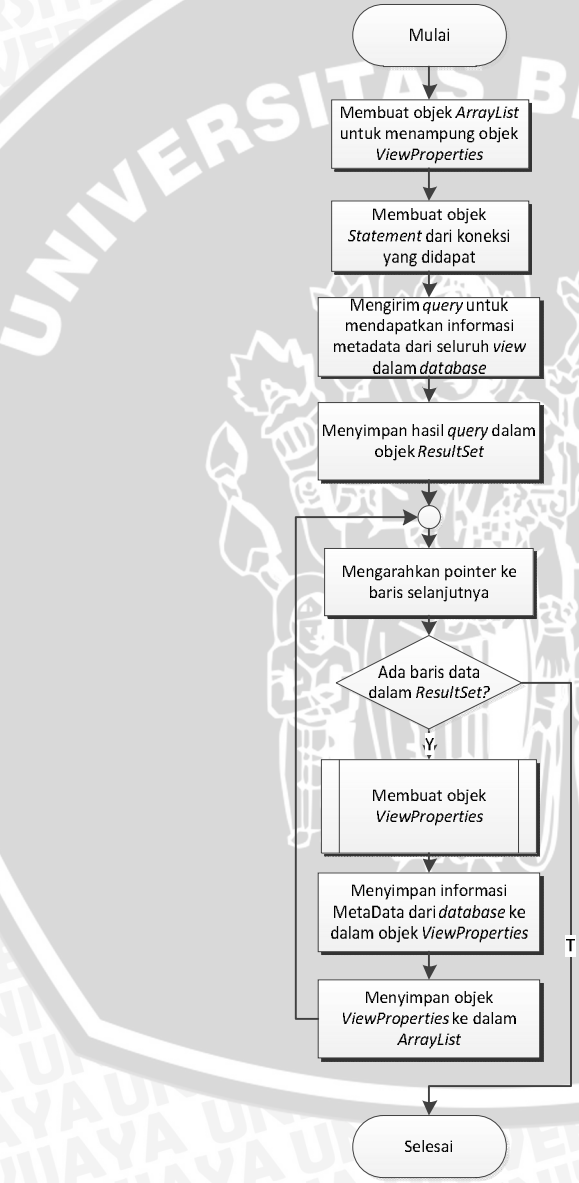
**c. PostgreSQL**

*Query* berikut berfungsi untuk mengambil informasi metadata *table name, engine name,* dan *auto increment* dari *information_schema.tables* dimana tabel

bertipe *base table* dan tabel bukan berasal dari *pg_catalog* atau *information_schema*:

```
"SELECT table_name FROM information_schema.tables WHERE table_type =
'BASE TABLE' AND table_schema NOT IN ('pg_catalog',
'information_schema');"
```

✓ *Method* **createViewList (Connection conn)**




**Gambar 4.9** *Flow Chart* Prosedur createViewList

Penjelasan diagram alir createViewList adalah sebagai berikut :

1. Membuat objek bertipe array untuk menampung objek-objek ViewProperties.
2. Membuat objek Statement dari koneksi yang didapat, untuk digunakan dalam pengiriman *query* dari API menuju DBMS tertentu.
3. Mengirim *query* untuk mengambil metadata dari suatu *view*.
4. Hasil *query* berupa objek ResultSet disimpan satu per satu kedalam objek ViewProperties.
5. Seluruh objek ViewProperties disimpan ke dalam objek array.

Untuk mendapatkan informasi metadata view dalam prosedur createViewList, maka dilakukan pengiriman *query* pada DBMS tujuan. *Query* yang dikirimkan pada masing-masing DBMS untuk mendapatkan informasi view yaitu :

**a. MySQL**

*Query* berikut berfungsi untuk mengambil informasi metadata *table name* dan *view_definition* dari *information_schema.VIEWS* yang dimiliki oleh *database* yang dijadikan parameter dalam variabel *dbname*:

```
"SELECT TABLE_NAME, VIEW_DEFINITION FROM information_schema.VIEWS
WHERE TABLE_SCHEMA = '"+dbName+"'"
```

**b. Oracle**

*Query* berikut berfungsi untuk mengambil informasi metadata *view name* dan *text* dari *user_views* :

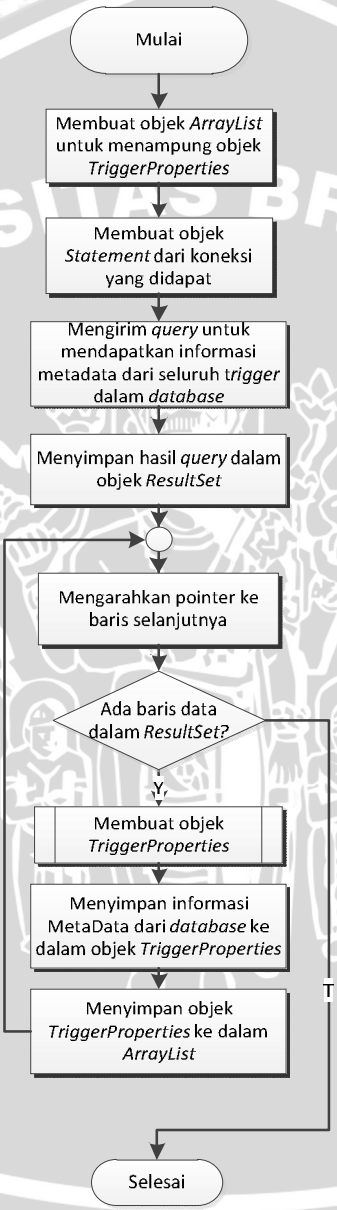
```
"SELECT VIEW_NAME, TEXT FROM user_views"
```

**c. PostgreSQL**

*Query* berikut berfungsi untuk mengambil informasi metadata *table name* dan *view_definition* dari *information_schema.views* dimana tabel bukan berasal dari *pg_catalog* atau *information_schema* dan nama tabel tidak diawali dengan ‘pg_’ :

```
"SELECT table_name, view_definition FROM information_schema.views
WHERE table_schema NOT IN ('pg_catalog', 'information_schema') AND
table_name !~ '^pg_';"
```

✓ *Method* **createTriggerList (Connection conn)**




**Gambar 4.10** *Flow Chart* Prosedur createTriggerList

Penjelasan diagram alir createTriggerList adalah sebagai berikut :

1. Membuat objek bertipe array untuk menampung objek-objek TriggerProperties.
2. Membuat objek Statement dari koneksi yang didapat, untuk digunakan dalam pengiriman *query* dari API menuju DBMS tertentu.
3. Mengirim *query* untuk mengambil metadata dari suatu *trigger*.
4. Hasil *query* berupa objek ResultSet disimpan satu per satu kedalam objek TriggerProperties.
5. Seluruh objek TriggerProperties disimpan ke dalam objek array.

Untuk mendapatkan informasi metadata *trigger* dalam prosedur createTriggerList, maka dilakukan pengiriman *query* pada DBMS tujuan. *Query* yang dikirimkan pada masing-masing DBMS untuk mendapatkan informasi *trigger* yaitu :

**a. MySQL**

*Query* berikut berfungsi untuk mengambil informasi metadata *trigger name, event manipulation, event object table, action statement, action orientation,* dan *action timing* dari *information_schema.TRIGGERS* yang dimiliki oleh *database* yang dijadikan parameter dalam variabel *dbname* :

```
"SELECT TRIGGER_NAME, EVENT_MANIPULATION, EVENT_OBJECT_TABLE,
ACTION_STATEMENT, ACTION_ORIENTATION, ACTION_TIMING FROM
information_schema.TRIGGERS WHERE TRIGGER_SCHEMA = '"+dbName+"'"
```

**b. Oracle**

*Query* berikut berfungsi untuk mengambil informasi metadata *trigger name*, *description*, dan *trigger body* dari *user_triggers* :

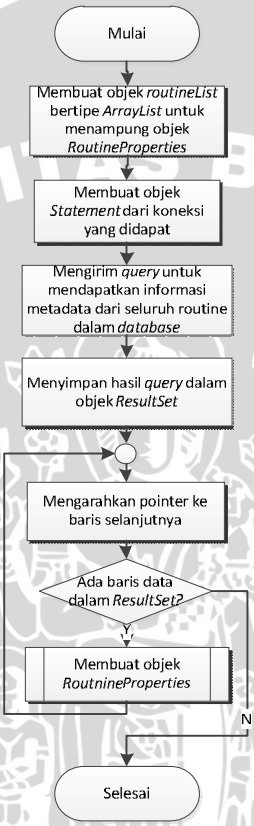
```
"SELECT trigger_name, description, trigger_body FROM user_triggers"
```

**c. PostgreSQL**

*Query* berikut berfungsi untuk mengambil informasi metadata *trigger name, event manipulation, event object table, action statement,* dan *condition timing* dari *information_schema.triggers* dimana tabel bukan berasal dari *pg_catalog* atau *information_schema* :

```
"SELECT trigger_name, event_manipulation, event_object_table,
action_statement, action_orientation, condition_timing FROM
information_schema.triggers WHERE trigger_schema NOT IN
('pg_catalog', 'information_schema');"
```

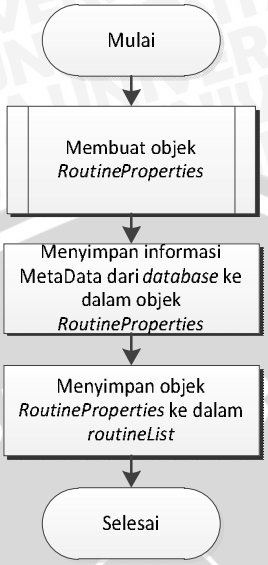
- ✓ *Method* **createRoutineList (Connection conn)**




**Gambar 4.11** *Flow Chart* Prosedur createRoutineList

Karena proses dalam pengambilan informasi metadata *routine* setiap vendor berbeda, maka berikut merupakan penjelasan gambar alur kerja (*flow chart*) prosedur / fungsi pengambilan informasi metadata *routine* setiap vendor :

✓ *Method* **Penyimpanan Informasi Routine MySQL**

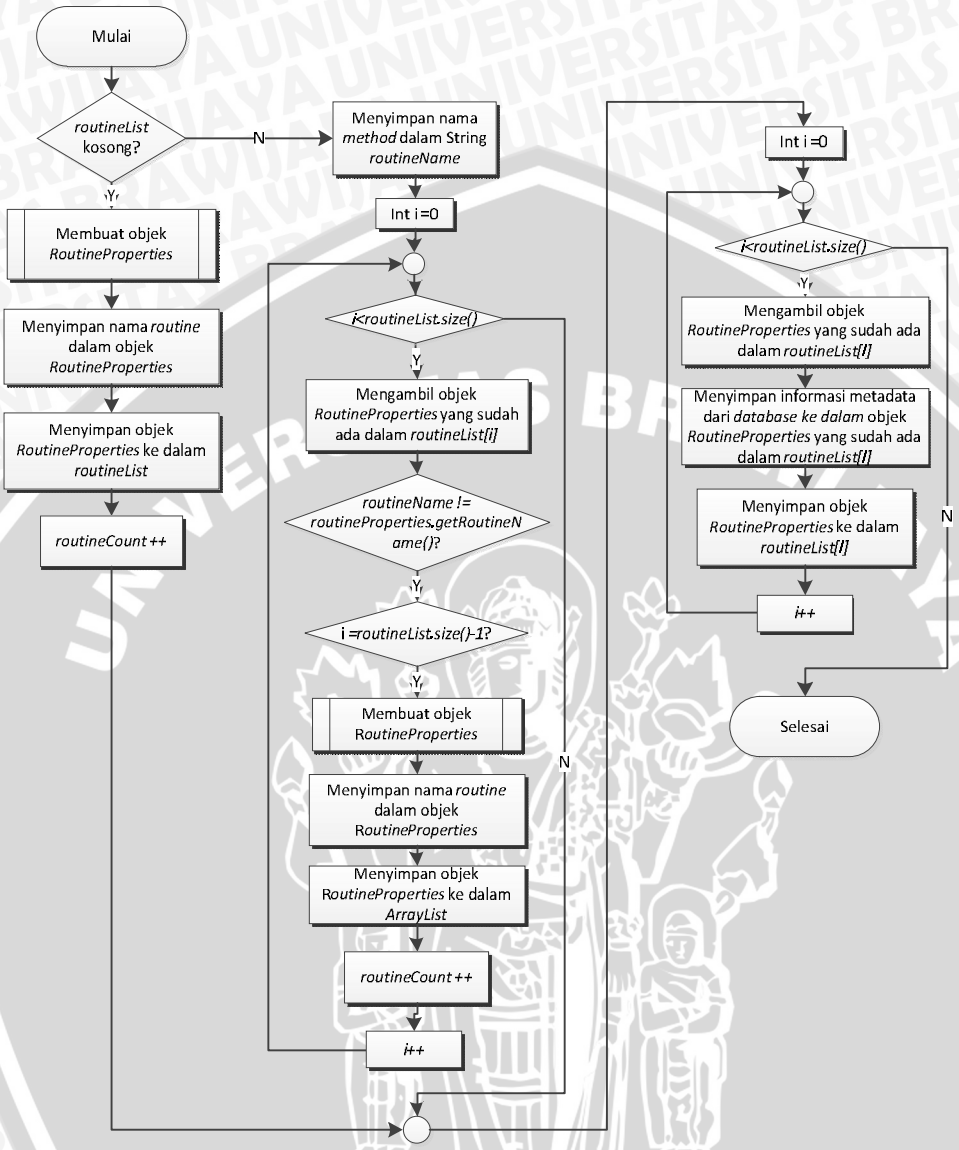



**Gambar 4.12** *Flow Chart* Prosedur Penyimpanan Informasi Routine MySQL

Penjelasan diagram alir prosedur penyimpanan informasi routine MySQL adalah sebagai berikut :

1. Membuat objek RoutineProperties.
2. Menyimpan informasi metadata dari *database* ke dalam objek RoutineProperties.
3. Menyimpan objek RoutineProperties ke dalam routineList.

✓ *Method* **Penyimpanan Informasi Routine Oracle**



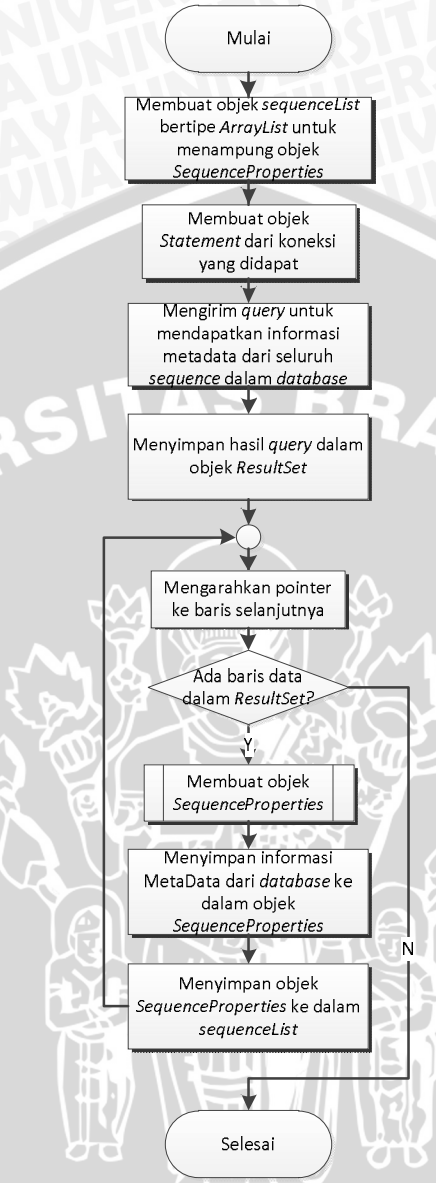

**Gambar 4.13** *Flow Chart* Prosedur Penyimpanan Informasi MetaData Oracle

Penjelasan diagram alir prosedur penyimpanan informasi metadata Oracle adalah sebagai berikut :

1. Memeriksa objek array routineList memiliki isi atau tidak.

2. Apabila routineList belum berisi objek RoutineProperties, maka informasi metadata *routine* dari *database* akan disimpan dalam objek RoutineProperties baru dan kemudian disimpan dalam routineList.
3. Apabila objek routineList telah berisi objek RoutineProperties, maka akan dilakukan proses pengecekan untuk setiap objek RoutineProperties yang ada dalam routineList dengan metadata *routine* baru yang di dapat dari *database*. Jika antara objek RoutineProperties dengan metadata *routine* baru memiliki nama *routine* yang sama, maka metadata *routine* baru akan melengkapi objek RoutineProperties yang telah ada. Jika tidak, maka akan dibuat objek RoutineProperties baru yang kemudian disimpan dalam routineList.

✓ *Method* **createSequenceList (Connection conn)**




**Gambar 4.14** *Flow Chart* Prosedur createSequenceList

Penjelasan diagram alir createSequenceList adalah sebagai berikut :

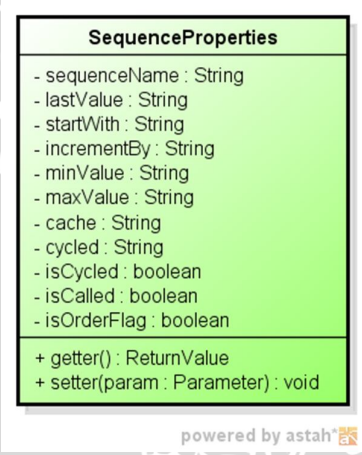
1. Membuat objek bertipe array untuk menampung objek-objek SequenceProperties.
2. Membuat objek Statement dari koneksi yang didapat, untuk digunakan dalam pengiriman *query* dari API menuju DBMS tertentu.

3. Mengirim *query* untuk mengambil metadata dari suatu *sequence.*
4. Hasil *query* berupa objek ResultSet disimpan satu per satu kedalam objek SequenceProperties.
5. Seluruh objek SequenceProperties disimpan ke dalam objek array.

#### 4.2.1.1.1 SequenceProperties

SequenceProperties merupakan kelas yang disediakan untuk menampung atribut yang dimiliki oleh fungsi *sequence* dalam *database* Oracle dan PostgreSQL, sedangkan MySQL tidak memiliki fungsi *sequence* karena dalam vendor ini tidak menggunakan fungsi *sequence* untuk melakukan *auto numbering*, melainkan menggunakan *Auto Increment* yang langsung menjadi atribut dari suatu kolom.

Gambar 4.15 menunjukkan *class diagram* dari kelas SequenceProperties :




**Gambar 4.15** SequenceProperties *Class Diagram*

Informasi metadata yang disimpan dalam atribut kelas ini antara lain :

- **sequenceName** : menyimpan nama *sequence.*
- **lastValue** : menyimpan nilai terakhir dari fungsi *sequence*.
- **startWith** : menyimpan nilai awal dari fingsi *sequence.*
- **incrementBy** : menyimpan nilai yang digunakan sebagai parameter kelipatan fungsi *sequence*.

- **minValue** : menyimpan nilai minimal yang dapat disimpan dalam fungsi *sequence*.
- **maxValue** : menyimpan nilai maksimal dalam fungsi *sequence*.
- **cache** : menyimpan nilai *cache* dari fungsi *sequence*.
- **cycled** : menyimpan nilai *cycle* dari fungsi *sequence*.
- **isCached** : menyimpan data *boolean* yang menunjukkan status *sequence* apakah menggunakan *cache* atau tidak.
- **isCycled** : menyimpan data *boolean* yang menunjukkan status *sequence* apakah menggunakan *cycle* atau tidak.
- **isOrderFlag** : menyimpan data *boolean* yang menunjukkan status *sequence* apakah menggunakan *order flag* atau tidak.

#### 4.2.1.1.2 TableProperties

TableProperties merupakan *interface* yang selanjutnya di implementasikan berdasarkan vendor tertentu menjadi kelas yang menyediakan fungsi-fungsi untuk mengambil dan menyimpan informasi metadata yang membentuk suatu tabel. Fungsi–fungsi tersebut digunakan oleh MetaDataProperties untuk mengambil informasi metadata (yang berkaitan dengan tabel) dari *database*. Informasi metadata tersebut meliputi nama tabel, nama engine (untuk MySQL), serta informasi metadata lain yang membentuk objek–objek baru. Objek–objek baru yang menyusun sebuah tabel tersebut antara lain :

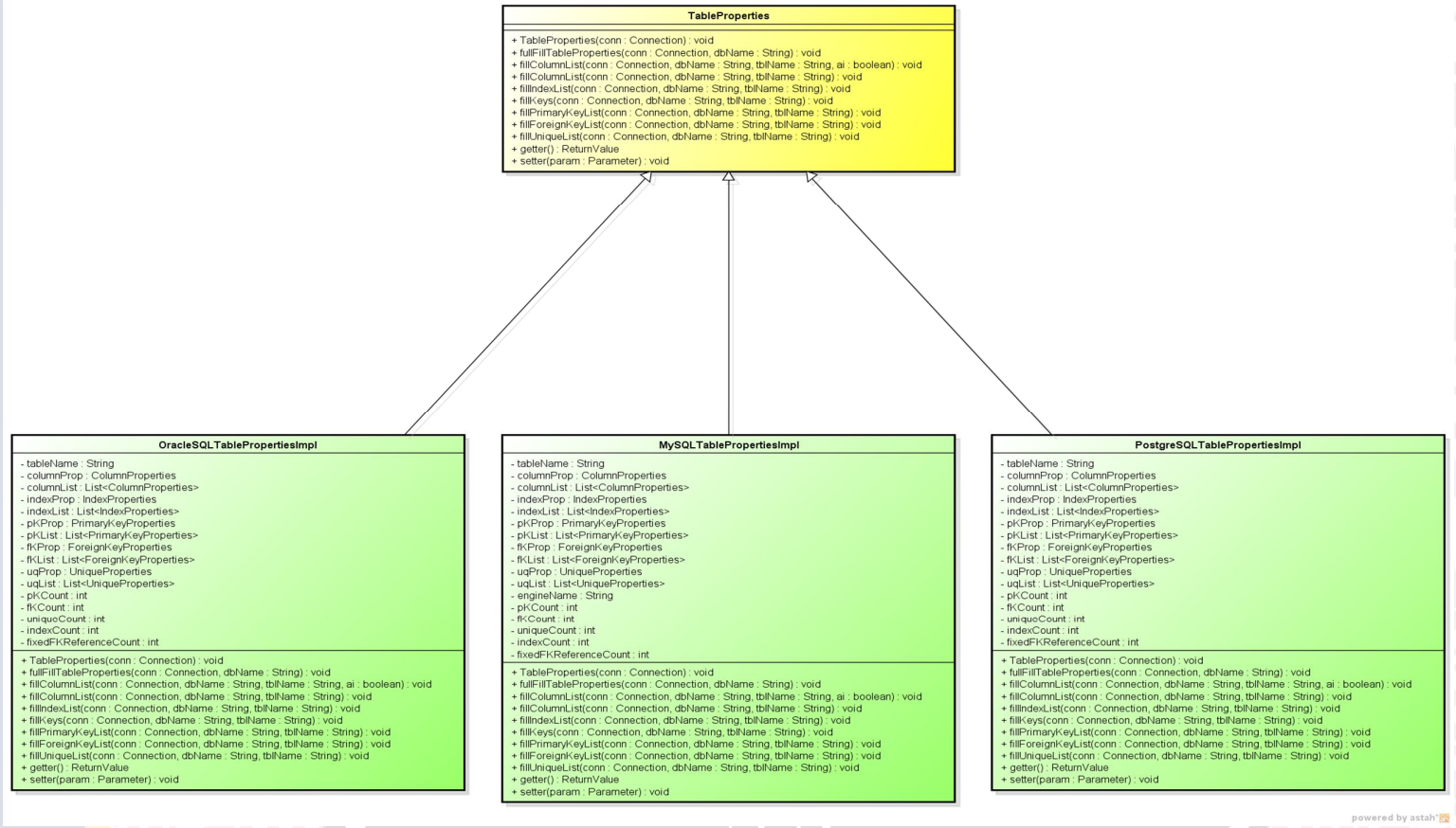
- ColumnProperties
- Indexes dan Keys
  - IndexProperties
  - PrimaryKeyProperties
  - UniqueProperties
  - ForeignKeyProperties
  - KeyParameterProperties

Dalam kelas ini terdapat pula beberapa atribut bertipe integer yang digunakan untuk menyimpan jumlah masing–masing objek yang tersimpan dalam

TableProperties, yang diantaranya adalah jumlah kolom dalam tabel, jumlah index dalam tabel,jumlah *primary key* dalam tabel, jumlah *unique key* dalam tabel, jumlah *foreign key* dalam tabel, serta disediakan satu buah atribut yang digunakan untuk membantu dalam proses pengurutan tabel–tabel referensional. Atribut ini digunakan sebagai parameter untuk mengetahui jumlah *foreign key* dalam tabel yang tabel referensinya sudah ditemukan dan diurutkan.

Untuk *database* MySQL, dibutuhkan atribut tambahan pada setiap objek tabel dimana atribut tersebut digunakan untuk menyimpan informasi tentang *auto numbering* (*Auto Increment*). Sedangkan Oracle dan PostgreSQL tidak membutuhkan atribut tersebut karena untuk melakukan *auto numbering*, Oracle dan PostgreSQL menggunakan fungsi *sequence*.

Gambar 4.16 menunjukkan *class diagram* dari TableProperties :




**Gambar 4.16** TableProperties *Class Diagram*

Informasi metadata yang disimpan dalam atribut kelas ini antara lain :

- **tableName** : menyimpan nama tabel.
- **engineName** : menyimpan nama *engine* yang digunakan oleh tabel.
- **isAutoIncrement** : menyimpan data boolean yang menunjukkan bahwa tabel memiliki id yang menggunakan *auto number* atau tidak.
- **nextAutoIncrement** : menyimpan nilai terakhir dari *auto number* suatu tabel.
- **columnProp** : menyimpan objek ColumnProperties yang berisi informasi metadata dari suatu kolom.
- **columnList** : menyimpan *list* seluruh objek ColumnProperties dalam objek TableProperties.
- **indexProp** : menyimpan objek IndexProperties yang berisi informasi metadata dari suatu index.
- **indexList** : menyimpan *list* seluruh objek IndexProperties dalam objek TableProperties.
- **pKProp** : menyimpan objek PrimaryKeyProperties yang berisi informasi metadata dari suatu *primary key*.
- **pKList** : menyimpan *list* seluruh objek PrimaryKeyProperties dalam objek TableProperties.
- **uniqueProp** : menyimpan objek UniqueProperties yang berisi informasi metadata dari suatu *unique key*.
- **uniqueList** : menyimpan list seluruh objek UniqueProperties dalam *database.*
- **fKProp** : menyimpan objek ForeignKeyProperties yang berisi informasi metadata dari suatu *foreign key*.
- **fKList** : menyimpan list seluruh objek ForeignKeyProperties dalam objek *TableProperties.*
- **indexCount** : menyimpan data jumlah index dalam objek TableProperties.

- **pKCount** : menyimpan data jumlah *primary key* dalam objek TableProperties.
- **fKCount** : menyimpan data jumlah *foreign key* dalam objek TableProperties.
- **uniqueCount** : menyimpan data jumlah *unique key* dalam objek TableProperties.
- **fixedFKReferenceCount** : menyimpan data jumlah *foreign key* dalam objek TableProperties yang tabel referensinya sudah ditemukan dan diurutkan.

Gambar 4.17 – 4.22 menunjukkan alur kerja (*flow chart*) fungsi – fungsi utama dalam kelas TableProperties :

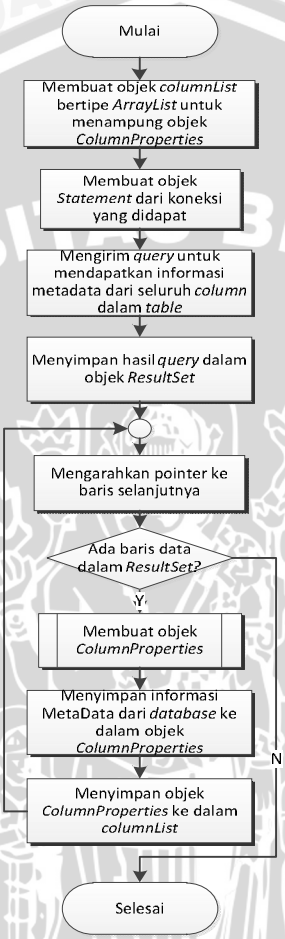
✓ *Constructor* **TableProperties(Connection conn)**


**Gambar 4.17** *Flow Chart Constructor* TableProperties

Fungsi utama dari *Constructor* TableProperties adalah untuk mempersiapkan objek-objek bertipe List, yang akan digunakan dalam pengelompokan dan

penyimpanan objek kolom, objek index, objek *primary key*, objek *foreign key* dan objek *unique key*.

- ✓ *Method* **fillColumnList(Connection conn, String dbName, String tblName, boolean ai)**




**Gambar 4.18** *Flow Chart* Prosedur fillColumnList

Penjelasan diagram alir fillColumnList adalah sebagai berikut :

1. Membuat objek bertipe array untuk menampung objek-objek ColumnProperties.
2. Membuat objek Statement dari koneksi yang didapat, untuk digunakan dalam pengiriman *query* dari API menuju DBMS tertentu.
3. Mengirim *query* untuk mengambil metadata dari suatu tabel.

4. Hasil *query* berupa objek ResultSet disimpan satu per satu kedalam objek ColumnProperties.
5. Seluruh objek ColumnProperties disimpan ke dalam objek array.

Untuk mendapatkan informasi metadata kolom dalam prosedur createColumnList, maka dilakukan pengiriman *query* pada DBMS tujuan. *Query* yang dikirimkan pada masing-masing DBMS untuk mendapatkan informasi kolom yaitu :

**a. MySQL**

Query berikut berfungsi untuk mengambil informasi metadata *column name, column default, is nullable, data type,character maximum length, column type, extra, character set name,* dan *collation name* dari *information_schema.COLUMNS* yang dimiliki oleh *database* dan *table* yang dijadikan parameter dalam variabel *dbname* dan *tblname* :

```
"SELECT COLUMN_NAME, COLUMN_DEFAULT, IS_NULLABLE, DATA_TYPE,
CHARACTER_MAXIMUM_LENGTH, COLUMN_TYPE, EXTRA, CHARACTER_SET_NAME,
COLLATION_NAME FROM information_schema.COLUMNS WHERE TABLE_SCHEMA =
'"+dbName+"' AND TABLE_NAME = '"+tblName+"'"
```

**b. Oracle**

Query berikut berfungsi untuk mengambil informasi metadata *column name, data type, data length, nullable, data default, data precision,* dan *data scale* dari *user_tab_columns* yang dimiliki oleh *table* yang dijadikan parameter dalam variabel *tableName* :

```
"SELECT COLUMN_NAME, DATA_TYPE, DATA_LENGTH, NULLABLE, DATA_DEFAULT,
DATA_PRECISION, DATA_SCALE FROM user_tab_columns WHERE
table_name='"+tableName+"' ORDER BY column_id"
```

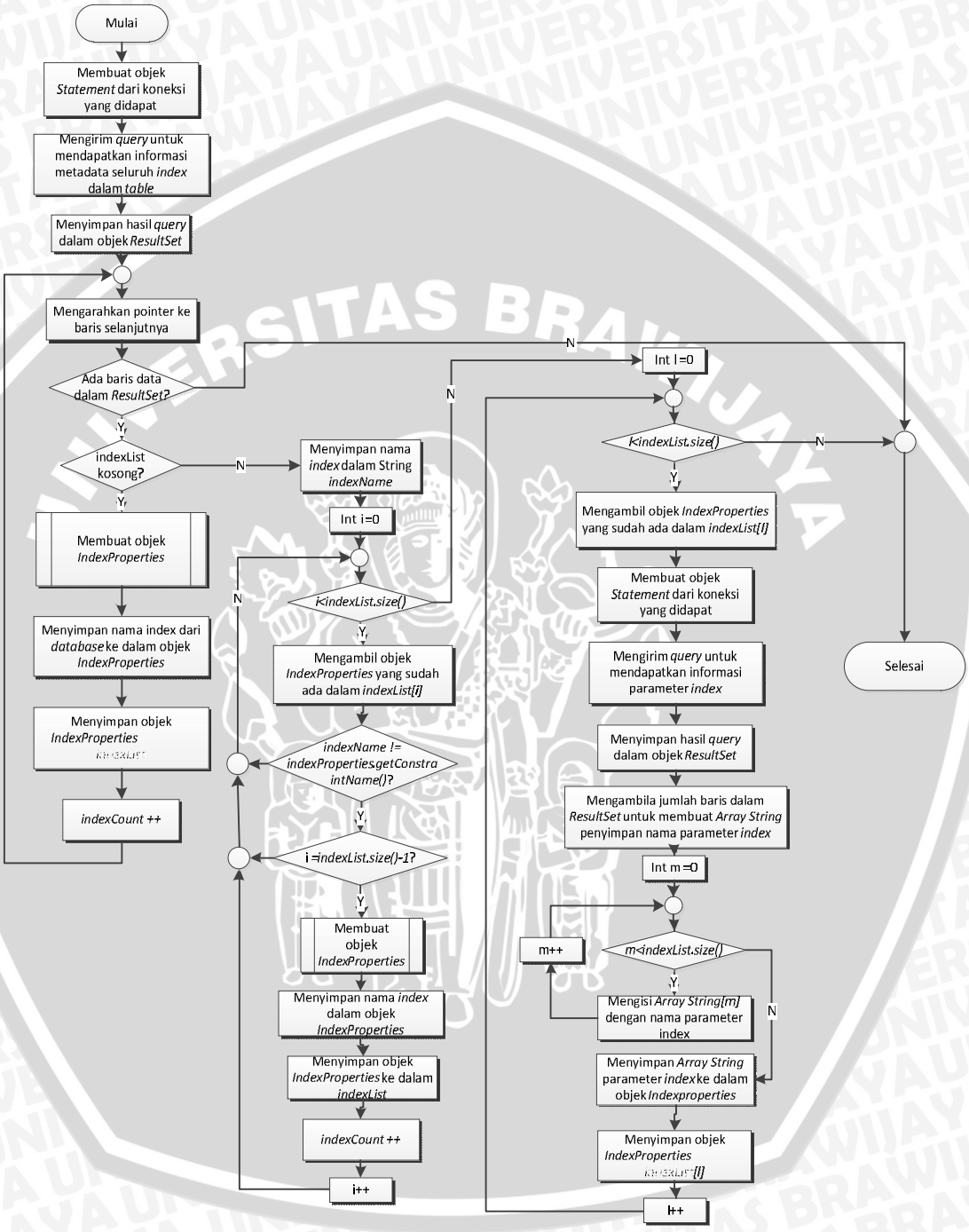
**c. PostgreSQL**

Query berikut berfungsi untuk mengambil informasi metadata *column name, ordinal position, data type, column default, is nullable, character maximum*

*length, numeric precision* dan *numeric scale* dari *information_schema.columns* yang dimiliki oleh *table* yang dijadikan parameter dalam variabel *tableName* :

```
"SELECT ordinal_position, column_name, data_type, column_default, is_nullable, character_maximum_length, numeric_precision, numeric_scale FROM information_schema.columns WHERE table_name = '"+tableName+"' ORDER BY ordinal_position;"
```

✓ *Method* **fillIndexList(Connection conn, String dbName, String tblName)**

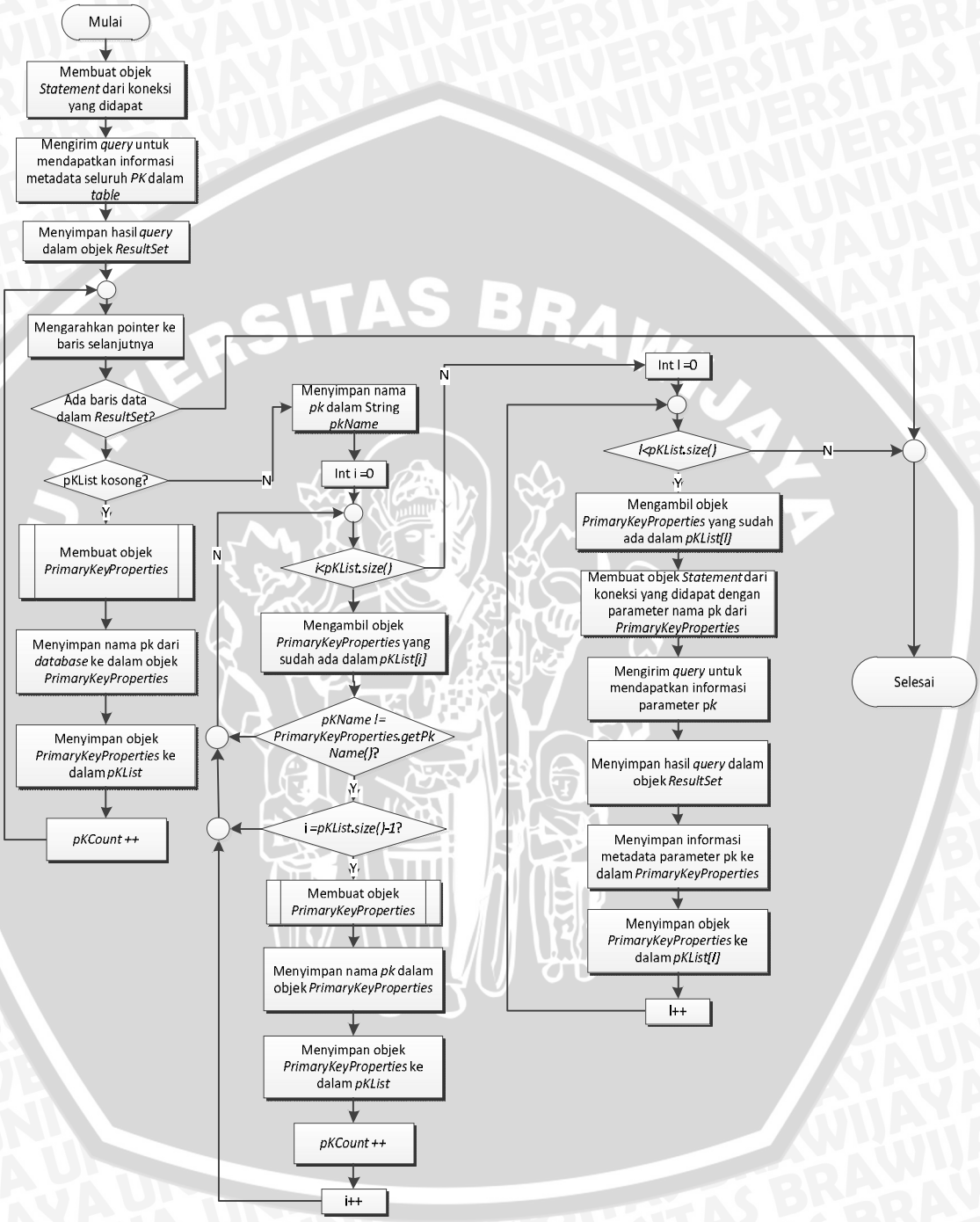



**Gambar 4.19** *Flow Chart* Prosedur fillIndexList

Penjelasan diagram alir fillIndexList adalah sebagai berikut :

1. Membuat objek Statement dari koneksi yang didapat, untuk digunakan dalam pengiriman *query* dari API menuju DBMS tertentu.
2. Mengirim *query* untuk mengambil informasi metadata index dari suatu tabel.
3. Hasil *query* berupa objek ResultSet disimpan kedalam objek IndexProperties.
4. Proses penyimpanan baris informasi dalam ResultSet menuju objek IndexProperties tidak dilakukan satu per satu, melainkan disimpan berdasarkan nama dari indexnya. Yang dimaksud penyimpanan berdasarkan nama index adalah ketika terdapat lebih dari satu baris informasi index dalam ResultSet yang memiliki nama index sama, maka informasi-informasi ini akan disimpan dalam satu objek IndexProperties.
5. Seluruh objek IndexProperties disimpan ke dalam objek array.

✓ *Method* **fillPrimaryKeyList(Connection conn, String dbName, String tblName)**

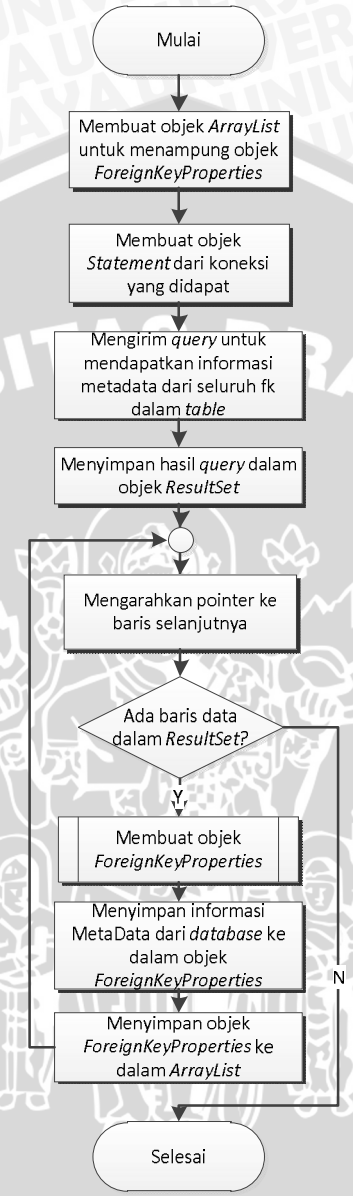



**Gambar 4.20** Flow Chart Prosedur fillPKList

Penjelasan diagram alir fillPKList adalah sebagai berikut :

1. Membuat objek Statement dari koneksi yang didapat, untuk digunakan dalam pengiriman *query* dari API menuju DBMS tertentu.
2. Mengirim *query* untuk mengambil informasi metadata *primary key* dari suatu tabel.
3. Hasil *query* berupa objek ResultSet disimpan kedalam objek PrimaryKeyProperties.
4. Proses penyimpanan baris informasi dalam ResultSet menuju objek PrimaryKeyProperties tidak dilakukan satu per satu, melainkan disimpan berdasarkan nama dari indexnya. Yang dimaksud penyimpanan berdasarkan nama index adalah ketika terdapat lebih dari satu baris informasi index dalam ResultSet yang memiliki nama index sama, maka informasi-informasi ini akan disimpan dalam satu objek PrimaryKeyProperties.
5. Seluruh objek PrimaryKeyProperties disimpan ke dalam objek array.

- ✓ *Method* **fillForeignKeyList(Connection conn, String dbName, String tblName)**



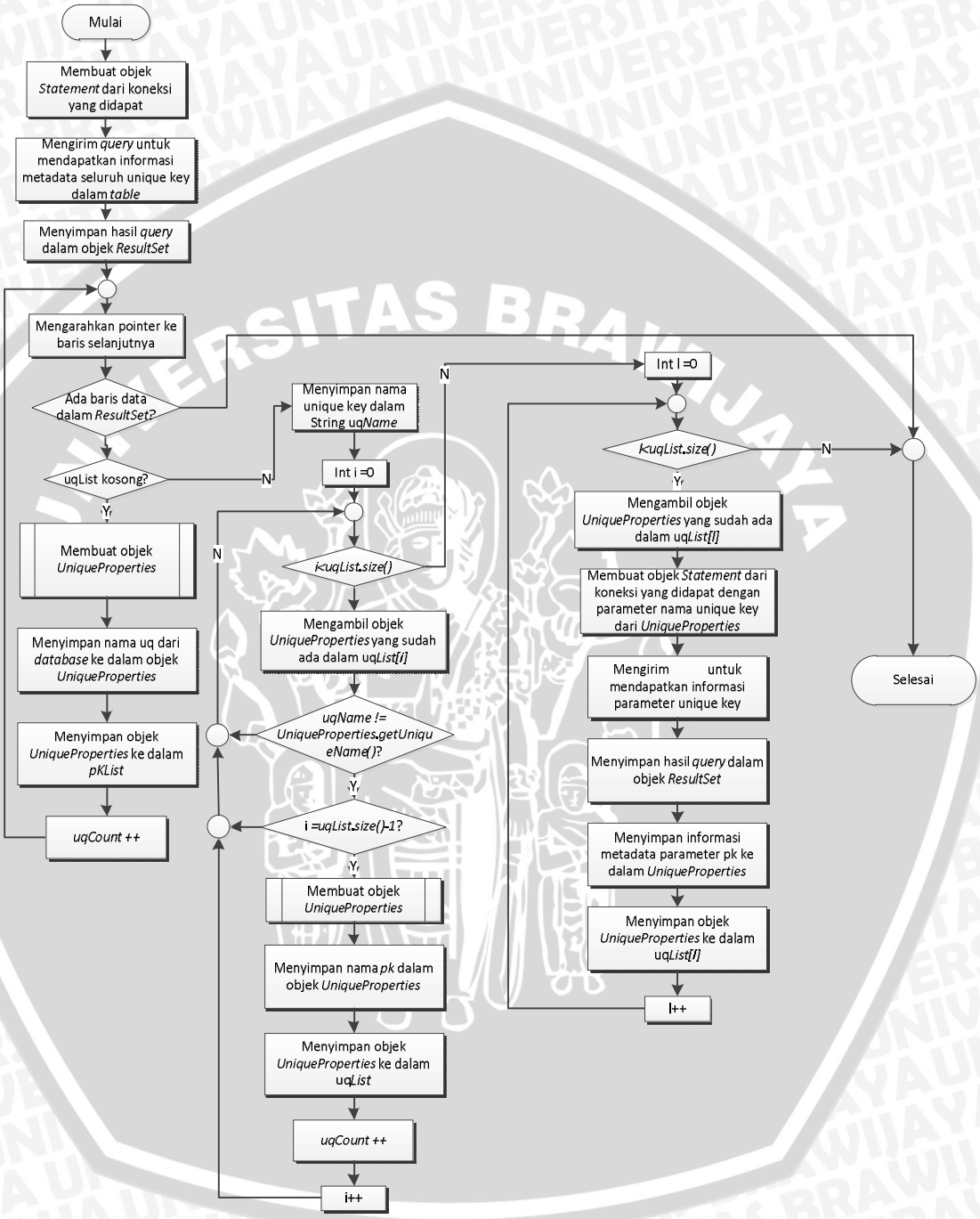

**Gambar 4.21** Flow Chart Prosedur fillFKList

Penjelasan diagram alir fillFKList adalah sebagai berikut :

1. Membuat objek bertipe array untuk menampung objek-objek ForeignKeyProperties.

2. Membuat objek Statement dari koneksi yang didapat, untuk digunakan dalam pengiriman *query* dari API menuju DBMS tertentu.
3. Mengirim *query* untuk mengambil informasi metadata *foreign key* dari suatu tabel.
4. Hasil *query* berupa objek ResultSet disimpan satu per satu kedalam objek ForeignKeyProperties.
5. Seluruh objek ForeignKeyProperties disimpan ke dalam objek array.

✓ *Method* **fillUniqueList(Connection conn, String dbName, String tblName)**



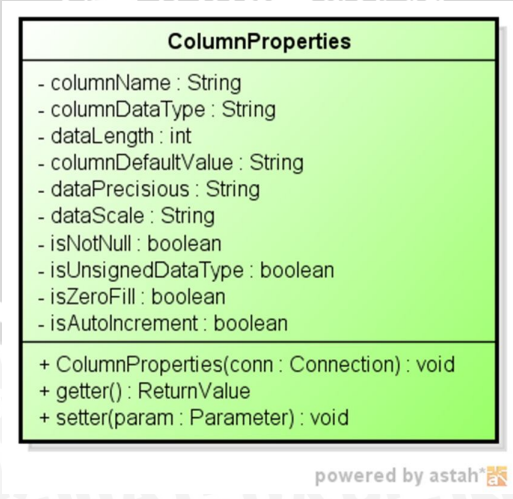

**Gambar 4.22** Flow Chart Prosedur fillUniqueList

Penjelasan diagram alir fillUniqueList adalah sebagai berikut :

1. Membuat objek Statement dari koneksi yang didapat, untuk digunakan dalam pengiriman *query* dari API menuju DBMS tertentu.
2. Mengirim *query* untuk mengambil informasi metadata *unique key* dari suatu tabel.
3. Hasil *query* berupa objek ResultSet disimpan kedalam objek UniqueKeyProperties.
4. Proses penyimpanan baris informasi dalam ResultSet menuju objek UniqueKeyProperties tidak dilakukan satu per satu, melainkan disimpan berdasarkan nama *unique key*-nya. Yang dimaksud penyimpanan berdasarkan nama *unique key* adalah ketika terdapat lebih dari satu baris informasi *unique key* dalam ResultSet yang memiliki nama *unique key* sama, maka informasi-informasi ini akan disimpan dalam satu objek UniqueKeyProperties.
5. Seluruh objek UniqueKeyProperties disimpan ke dalam objek array.

#### 4.2.1.1.2.1 ColumnProperties

ColumnProperties merupakan kelas yang disediakan untuk menampung atribut yang dimiliki oleh setiap kolom dalam tabel. Gambar 4.23 menunjukkan *class diagram* dari kelas ColumnProperties :



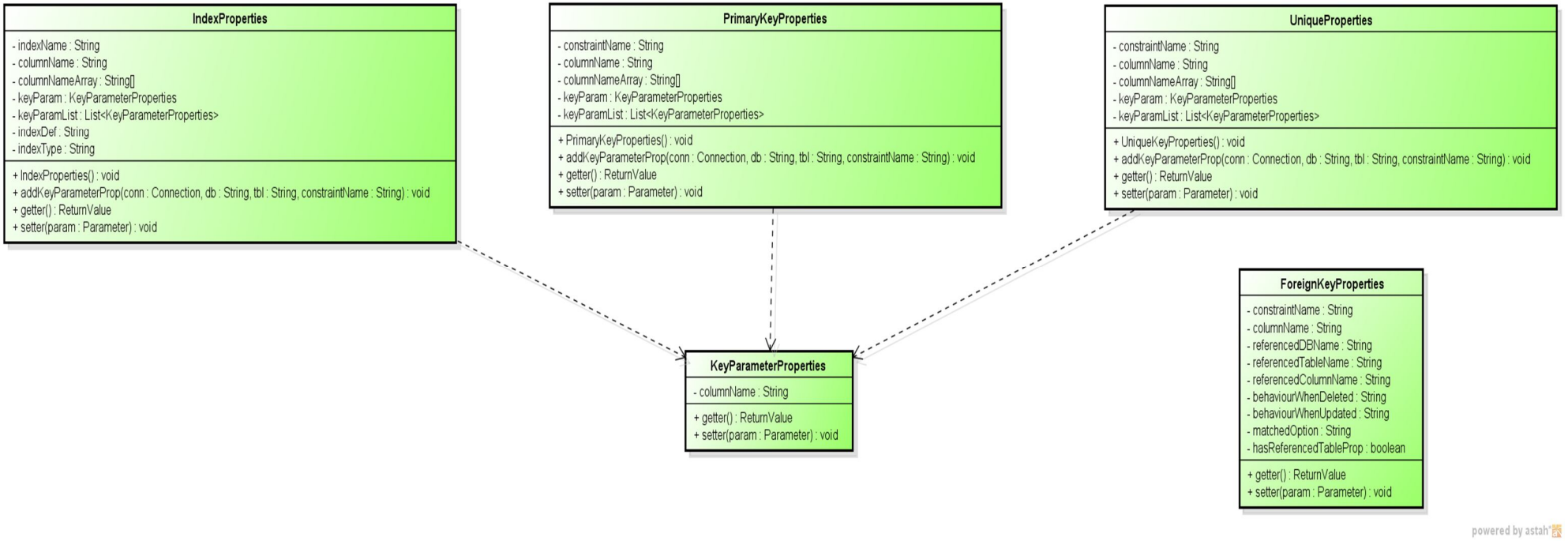

**Gambar 4.23** ColumnProperties *Class Diagram*

Informasi metadata yang disimpan dalam atribut kelas ini antara lain :

- **columnName** : menyimpan nama kolom.
- **columnDataType** : menyimpan tipe data dari kolom.
- **dataLength** : menyimpan lebar dari tipe data kolom.
- **columnDefaultValue** : menyimpan nilai *default* yang digunakan oleh kolom.
- **dataPrecisious** : menyimpan nilai presisi dari tipe data kolom.
- **dataScale** : menyimpan nilai skala dari tipe data kolom.
- **isNotNull** : menyimpan data *boolean* yang menunjukkan bahwa kolom boleh berisi null atau tidak.
- **isUnsignedDataType** : menyimpan data *boolean* yang menunjukkan bahwa tipe data kolom adalah tipe data *unsigned.*
- **isZeroFill** : menyimpan data boolean yang menunjukkan bahwa ketika kolom tidak diisi, maka kolom akan secara *default* bernilai "0".
- **isAutoIncrement** : menyimpan data *boolean* yang menunjukkan bahwa kolom berisi data *auto number*.

#### 4.2.1.1.2.2 Index dan Keys

Untuk menciptakan index dan *keys* (*primary key*, *unique key*, dan *foreign key*) pada suatu tabel, dibutuhkan parameter nama kolom yang menjadi letak index dan *keys* tersebut diimplementasikan. Gambar 4.24 menunjukkan *class diagram* dari objek index dan keys :




**Gambar 4.24** Index dan Keys *Class Diagram*

#### 4.2.1.1.2.2.1 IndexProperties

IndexProperties merupakan kelas yang menyediakan atribut-atribut dan fungsi –fungsi untuk mengambil dan menyimpan informasi metadata yang berkaitan dengan index dalam *database*. Informasi metadata index yang disimpan dalam kelas IndexProperties antara lain :

- **indexName** : merupakan nama dari index.
- **columnName** : merupakan nama kolom yang menjadi parameter index.
- **columnNameArra**y : merupakan kumpulan nama kolom yang menjadi parameter index, digunakan ketika suatu index memiliki parameter lebih dari satu kolom.
- **keyParam** : merupakan objek yang menyimpan nama kolom dan menjadi parameter index.
- **keyParamList** : merupakan kumpulan objek yang menyimpan nama kolom dan menjadi parameter index, digunakan ketika suatu index memiliki parameter lebih dari satu kolom.
- **indexDef** : merupakan definisi suatu index.
- **indexType** : merupakan pengkhususan tipe dari suatu index.

#### 4.2.1.1.2.2.2 PrimaryKeyProperties

PrimaryKeyProperties merupakan kelas yang menyediakan atribut-atribut dan fungsi –fungsi untuk mengambil dan menyimpan informasi metadata yang berkaitan dengan primary key dalam *database*. Informasi metadata *primary key* yang disimpan antara lain :

- **constraintName** : merupakan nama dari *primary key*.
- **columnName** : merupakan nama kolom yang menjadi parameter *primary key*.

- **columnNameArray** : merupakan kumpulan nama kolom yang menjadi parameter *primary key*, digunakan ketika suatu *primary key* memiliki parameter lebih dari satu kolom.
- **keyParam** : merupakan objek yang menyimpan nama kolom dan menjadi parameter *primary key*.
- **keyParamList** : merupakan kumpulan objek yang menyimpan nama kolom dan menjadi parameter *primary key*, digunakan ketika suatu *primary key* memiliki parameter lebih dari satu kolom.

#### 4.2.1.1.2.2.3 UniqueProperties

UniqueProperties merupakan kelas yang menyediakan atribut-atribut dan fungsi –fungsi untuk mengambil dan menyimpan informasi metadata yang berkaitan dengan *unique key* dalam *database*. Informasi metadata *unique key* yang disimpan antara lain :

- **constraintName** : merupakan nama dari *unique key*.
- **columnName** : merupakan nama kolom yang menjadi parameter *unique key*.
- **columnNameArray** : merupakan kumpulan nama kolom yang menjadi parameter *unique key*, digunakan ketika suatu *unique key* memiliki parameter lebih dari satu kolom.
- **keyParam** : merupakan objek yang menyimpan nama kolom dan menjadi parameter *unique key*.
- **keyParamList** : merupakan kumpulan objek yang menyimpan nama kolom dan menjadi parameter *unique key*, digunakan ketika suatu *unique key* memiliki parameter lebih dari satu kolom.

#### 4.2.1.1.2.2.4 ForeignKeyProperties

ForeignKeyProperties merupakan kelas yang menyediakan atribut-atribut dan fungsi –fungsi untuk mengambil dan menyimpan informasi metadata yang berkaitan dengan *foreign key* dalam *database*. Informasi metadata *foreign key* yang disimpan antara lain :

- **constraintName** : merupakan nama dari *foreign key*.
- **columnName** : merupakan nama kolom yang menjadi parameter *foreign key*.
- **referencedDBName :** merupakan nama *database* yang direferensi oleh *foreign key*.
- **referencedTableName** : merupakan nama tabel yang direferensi oleh parameter *foreign key*.
- **referencedColumnName** : merupakan nama kolom yang direferensi oleh parameter *foreign key*.
- **behaviourWhenDeleted** : merupakan suatu kondisi yang dilakukan ketika data yang direferensi oleh parameter *foreign key* dihapus.
- **behaviourWhenUpdated** : merupakan suatu kondisi yang dilakukan ketika data yang direferensi oleh parameter *foreign key* diperbarui.
- **hasReferencedTableProp** : merupakan data *boolean* yang menandakan bahwa tabel yang direferensi oleh objek *foreign key* telah ditemukan, atribut ini dignakan ketika melakukan *sorting* tabel – tabel relasional.

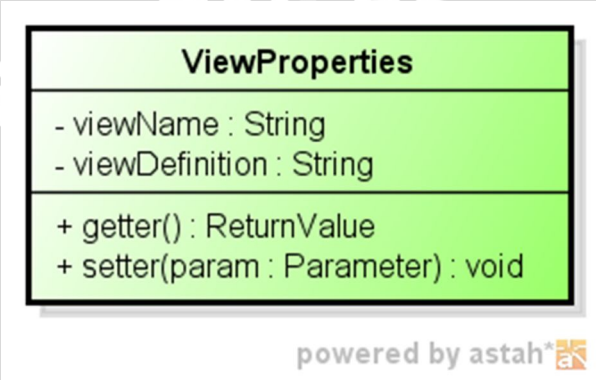
#### 4.2.1.1.2.2.5 KeyParameterProperties

KeyParameterProperties merupakan kelas yang digunakan untuk menyimpan parameter dari index / *keys*. Parameter yang dimaksud berupa nama kolom dimana index / *keys* diimplementasikan didalamnya. Informasi metadata yang disimpan dalam atribut kelas ini yaitu :

- **columnName :** merupakan nama kolom yang menjadi parameter dari index / *keys*.

#### 4.2.1.1.3 ViewProperties

ViewProperties merupakan kelas yang disediakan untuk menampung atribut yang dimiliki oleh setiap tabel. Gambar 4.25 menunjukkan *class diagram* dari kelas ViewProperties :



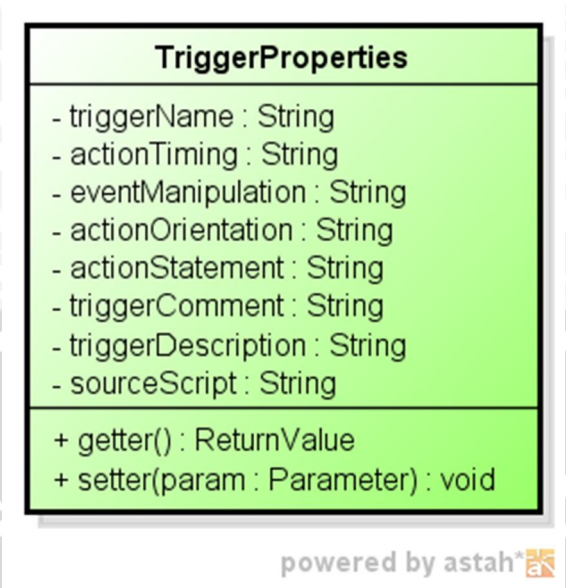

**Gambar 4.25** ViewProperties *Class Diagram*

Informasi metadata yang disimpan dalam atribut kelas ini yaitu :

- **viewName** : merupakan nama dari *view*.
- **viewDefinition** : merupakan definisi dari *view* yang berisi *query* struktur *view*.

#### 4.2.1.1.4 TriggerProperties

TriggerProperties merupakan kelas yang disediakan untuk menampung atribut yang dimiliki oleh setiap tabel. Gambar 4.26 menunjukkan *class diagram* dari kelas TriggerProperties :



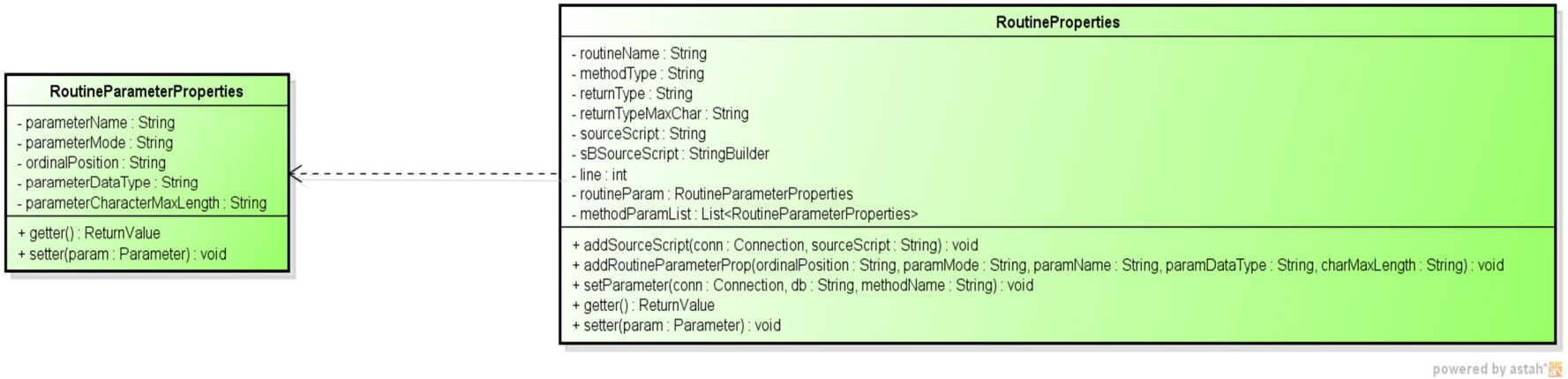

**Gambar 4.26** TriggerProperties *Class Diagram*

Informasi metadata yang disimpan dalam atribut kelas ini yaitu :

- **triggerName** : merupakan nama dari *trigger.*
- **actionTiming** : merupakan keterangan waktu aksi *trigger* (AFTER dan BEFORE ) terhadap *event* yang menjadi parameternya.
- **eventManipulation** : merupakan keterangan kejadian yang menjadi parameter *trigger*.
- **actionOrientation** : merupakan keterangan orientasi dari aksi yang dilakukan oleh *trigger*.
- **triggerDescription** : meyimpan isi kepala *query* dari *trigger* (Oracle).
- **triggerComment :** menyimpan isi badan *query* dari *trigger* (Oracle).
- **sourceScript** : menyimpan *query* dari *trigger*.

#### 4.2.1.1.5 RoutineProperties

RoutineProperties merupakan kelas yang menyediakan fungsi-fungsi untuk mengambil dan menyimpan informasi metadata yang membentuk suatu metode (*stored procedure* atau *stored function*). Fungsi – fungsi tersebut digunakan oleh MetaDataProperties untuk mengambil informasi metadata suatu metode dari *database*. Gambar 4.27 menunjukkan *class diagram* dari kelas RoutineProperties beserta kelas yang menyimpan parameter dari metode tersebut :




**Gambar 4.27** RoutineProperties *Class Diagram*

Informasi metadata yang disimpan dalam atribut kelas ini antara lain :

- **routineName** : merupakan nama dari metode.
- **routineType** : merupakan jenis dari metode (prosedur atau fungsi).
- **returnType** : merupakan tipe data kembalian dari metode.
- **returnTypeMaxChar** : merupakan jumlah maksimal karakter kembalian dari metode.
- **sourceScript :** meyimpan *syntax* dari metode.
- **sBSourceScript** : meyimpan gabungan *syntax* dari metode yang dimiliki Oracle (Oracle menyimpan *source script* dari metode dengan cara memecahnya menjadi beberapa baris tertentu).
- **line** : menyimpan nomor baris dari metode (Oracle).
- **routineParam** : menampung objek parameter yang dimiliki oleh metode tersebut.
- **routineParamList** : menyimpan list objek parameter yang dimiliki oleh metode tersebut.

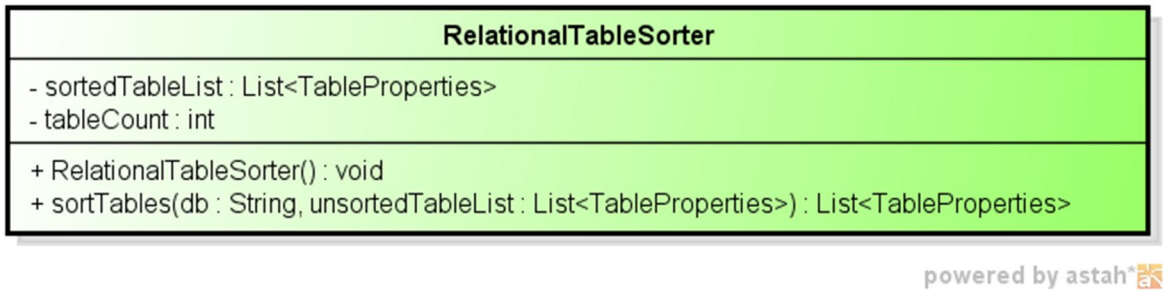
#### 4.2.1.1.5.1 RoutineParameterProperties

RoutineParameterProperties merupakan kelas yang digunakan untuk menyimpan parameter dari metode. Informasi metadata yang disimpan dalam atribut kelas ini antara lain :

- **parameterName** : merupakan nama parameter dari metode.
- **parameterMode** : merupakan parameter mode dari metode.
- **ordinalPosition** : merupakan nomor urut parameter.
- **parameterDataType** : merupakan tipe data parameter.
- **parameterCharacterMaxLength** : merupakan jumlah maksimal karakter dari parameter

#### 4.2.1.1.6 RelationalTableSorter

RelationalTableSorter merupakan kelas yang menyediakan fungsi – fungsi untuk digunakan dalam pengurutan tabel – tabel relasional. Dengan menggunakan fungsi dalam kelas ini, maka objek – objek tabel dapat tersusun mulai dari tabel – tabel yang menjadi master (tabel utama yang tidak memiliki referensi ke tabel lain), hingga tabel – tabel yang harus mereferensi tabel lain. Gambar 4.28 menunjukkan *class diagram* dari kelas RelationalTableSorter beserta kelas yang menyimpan parameter dari metode tersebut :



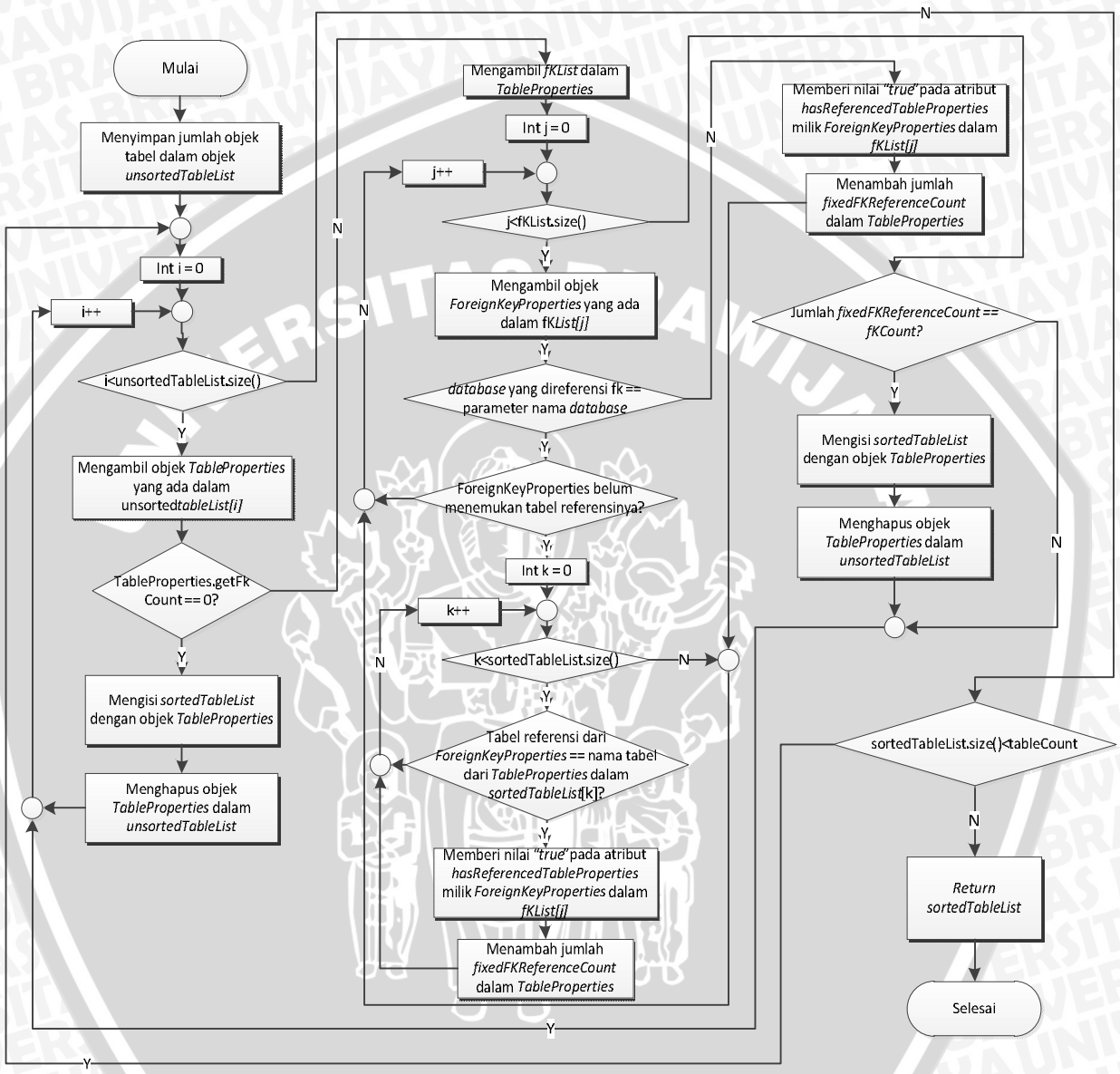

**Gambar 4.28** RelationalTableSorter *Class Diagram*

Informasi metadata yang disimpan dalam atribut kelas ini antara lain :

- **storedTableList** : menyimpan list dari objek-objek TableProperties yang sudah diurutkan berdasarkan relasinya.
- **tableCount** : merupakan jumlah tabel yang ada dalam list dan digunakan sebagai parameter pengontrol dalam proses penyusunan tabel – tabel relasional.

Gambar 4.29 menunjukkan alur kerja (*flow chart*) fungsi – fungsi utama dalam kelas RoutineProperties :

✓ *Method* **sortTables (String db, List<TableProperties> unsortedTableProperties)**




**Gambar 4.29** *Flow Chart* Fungsi sortTables

Penjelasan diagram alir sortTables adalah sebagai berikut :

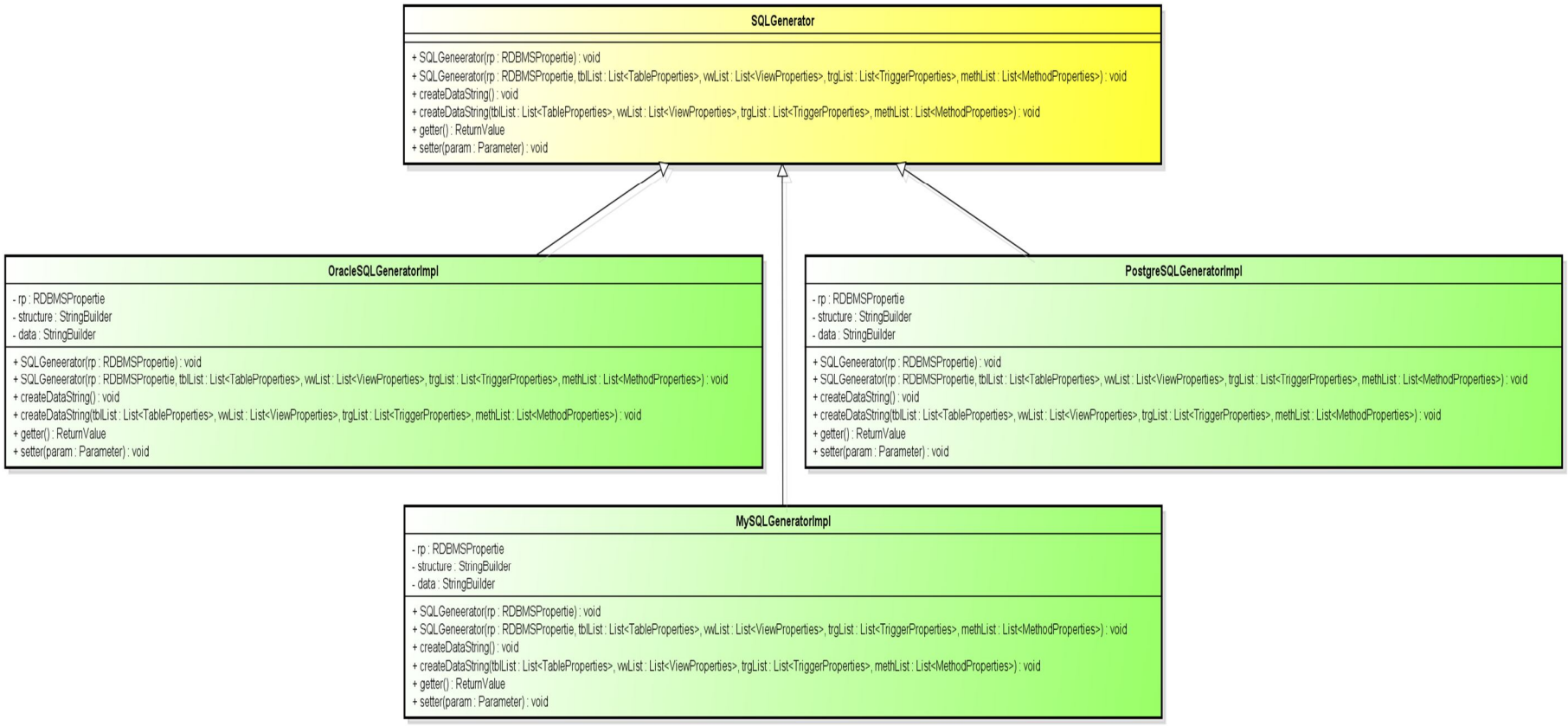
1. Menyimpan jumlah objek tabel yang ada dalam unsortedTableList ke dalam sebuah variabel.
2. Melakukan perulangan untuk melakukan pemeriksaan setiap objek TableProperties yang ada dalam unsortedTableList dengan menggunakan jumlah tabel yang didapat pada poin (1) sebagai parameter jumlah perulangan.
3. Pemeriksaan yang dilakukan adalah dengan memeriksa ada atau tidaknya *foreign key* dalam tabel tersebut. Apabila tidak ada *foreign key*, maka TableProperties dapat dipindahkan menuju objek sortedTableList.
4. Apabila suatu tabel memiliki *foreign key* , maka akan dilakukan pemeriksaan apakah seluruh tabel yang direferensi sudah ada dalam objek sortedTableList atau tidak. Bila seluruh tabel yang direferensi oleh *foreign key* sudah ada dalam objek sortedTableList, maka TableProperties tersebut akan dipindahkan ke dalam objek sortedTableList.
5. Apabila sebuah tabel masih memiliki *foreign key* yang tabel relasinya belum ditemukan, maka proses pemeriksaan akan terus berulang hingga seluruh tabel relasi dapat berpindah ke dalam objek sortedTableList. Hal ini berarti seluruh tabel relasi telah selesai diurutkan berdasarkan hubungan referensinya.

#### 4.2.1.2 SQLGenerator

SQLGenerator merupakan *interface* yang selanjutnya akan diimplementasikan menjadi kelas yang menyediakan fungsi – fungsi untuk melakukan *generate* SQL *script* berdasarkan vendornya. Implementasi SQLGenerator dibedakan berdasarkan vendor karena setiap vendor memiliki perbedaan dalam menuliskan *script*-nya.

Kelas SQLGenerator melakukan *generate* SQL *script* dengan menerima objek RDBMSProperties yang di dalam menyimpan objek – objek yang berisi seluruh informasi metadata dari *database*. Selanjutnya, informasi metadata tersebut diambil dan disusun sesuai kaidah penulisan SQL *script* berdasarkan vendornya.

Gambar 4.30 menunjukkan *interface* dan *class diagram* dari SQLGenerator beserta kelas yang menyimpan parameter dari metode tersebut :




**Gambar 4.30** SQLGenerator *Class Diagram*

Informasi metadata yang disimpan dalam atribut kelas ini antara lain :

- **rp** : menyimpan objek RDBMSPropperties.
- **structure** : menyimpan String SQL hasil dari proses *generate* struktur *database*.
- **data** : menyimpan String SQL hasil dari proses *generate* data di *database*.

Gambar 4.31 menunjukkan alur kerja (*flow chart*) fungsi – fungsi utama dalam kelas SQLGenerator :

✓ *Method* **createDataString()**

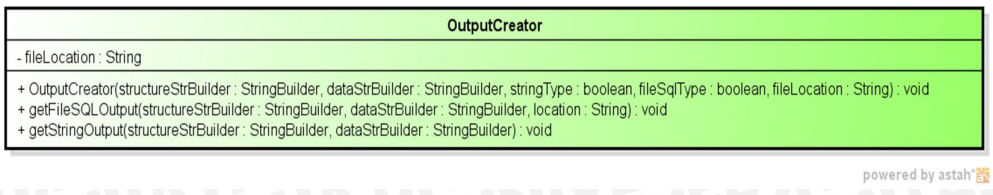

**Gambar 4.31** *Flow Chart* Prosedur createDataString

Penjelasan diagram alir createDataString adalah sebagai berikut :

1. Dalam fungsi createDataString disediakan sebuah kerangka SQL *script* yang disusun berdasarkan standar penulisan SQL *script.*
2. Proses penyusunan SQL *script* dilakukan secara berurutan, mulai dari :
    a. Penyusunan seluruh *sequence script* yang ada dalam objek sequenceList.
    b. Penyusunan seluruh *table script* yang ada dalam objek tableList. Penyusunan *table script* juga meliputi penyusunan seluruh atribut-atribut dari tabel yaitu *index script, primary key script, foreign key script,* dan *unique key script.*
    c. Penyusunan seluruh *view script* yang ada dalam objek viewList.
    d. Penyusunan seluruh *trigger script* yang ada dalam objek triggerList.
    e. Penyusunan seluruh *routine script* yang ada dalam objek routineList.
3. createDataString adalah proses pembentukan SQL *script* seutuhnya dengan cara penggabungan antara kerangka SQL dan variabel-variabel berisi informasi metadata yang telah didapatkan dari *database* tertentu.

### 4.2.2 OutputTool

OutputTool merupakan kelas yang menyediakan fungsi – fungsi untuk menghasilkan jenis keluaran SQL *script* (apakah akan dikeluarkan dalam bentuk String, atau disimpan dalam file .sql). Gambar 4.32 menunjukkan *class diagram* dari kelas OutputTool beserta kelas yang menyimpan parameter dari metode tersebut :



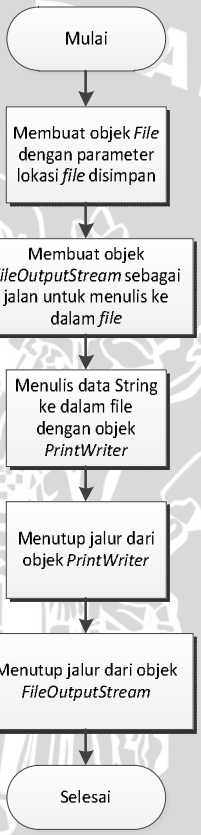

**Gambar 4.32** OutputTool *Class Diagram*

Parameter yang disimpan dalam kelas ini antara lain :

- fileLocation : menyimpan informasi lokasi penyimpanan file .sql.

Gambar 4.33 menunjukkan alur kerja (*flow chart*) fungsi – fungsi utama dalam kelas RoutineProperties :

✓ *Method* **getFileSQLOutput(StringBuilder strBuilder, StringBuilder dataStrBuilder, String location)**




**Gambar 4.33** *Flow Chart* Prosedur getFileSQLOutput

Penjelasan diagram alir getFileSQLOutput adalah sebagai berikut :

1. Membuat objek File dengan memberikan parameter lokasi *file* akan disimpan.
2. Membuat objek FileOutputStream sebagai jalan untuk melakukan penulisan ke dalam *file*.

3. Menulis data String ke dalam *file* dengan menggunakan objek PrintWriter.
4. Menutup jalur dari PrintWriter.
5. Menutup jalur dari FileOutputStream.

# BAB V
# PENGUJIAN

Untuk mengetahui apakah ToREn API dapat bekerja dengan baik dan sesuai dengan perancangan, maka diperlukan serangkaian pengujian. Pengujian yang dilakukan meliputi pembuatan sebuah aplikasi sederhana yang menggunakan ToREn API pada skripsi ini, sehingga aplikasi tersebut menjadi jembatan pembukti bahwa :

1. ToREn API dapat melakukan *reverse engineering database,* dengan menggunakan koneksi yang didapat, dari multi DBMS (MySQL, Oracle, PostgreSQL) menjadi SQL *script* dengan struktur *database* yang benar.
2. SQL *script* hasil *reverse engineering database* dari ToREn API dapat dieksekusi pada DBMS asalnya.
3. Kesimpulan hasil pengujian.

## 5.1 Pengujian *Reverse Engineering Database* menjadi SQL *Script*

Pengujian ini bertujuan untuk mengetahui apakah ToREn API yang telah diimplementasikan dalam sebuah aplikasi *reverse engineering database*, dapat menghasilkan SQL *script* sesuai dengan struktur *database* yang benar.

Pengujian dibagi menjadi tiga bagian, yang masing – masing bagian mewakili satu buah DBMS. Setiap pengujian satu DBMS, akan dilakukan *reverse engineering database* pada :

a. 1 buah tabel dengan kolom yang memiliki beberapa macam tipe data, *null constraint, default value, index, primary key, unique key*, dan data sebagai isi dari tabel.
b. 2 buah tabel yang berelasi dengan menggunakan *foreign key* dan data sebagai isi dari tabel-tabel.
c. 1 buah *view* dari hasil *query* tabel – tabel pada point (b.).
d. 1 buah *trigger.*
e. 2 buah *routines* yang terdiri dari 1 buah *stored procedure* dan 1 buah *stored function*

f. 1 buah *sequence function* (Oracle dan PostgreSQL).

### 5.1.1 Pengujian *Reverse Engineering Database* menjadi SQL *Script* DBMS MySQL

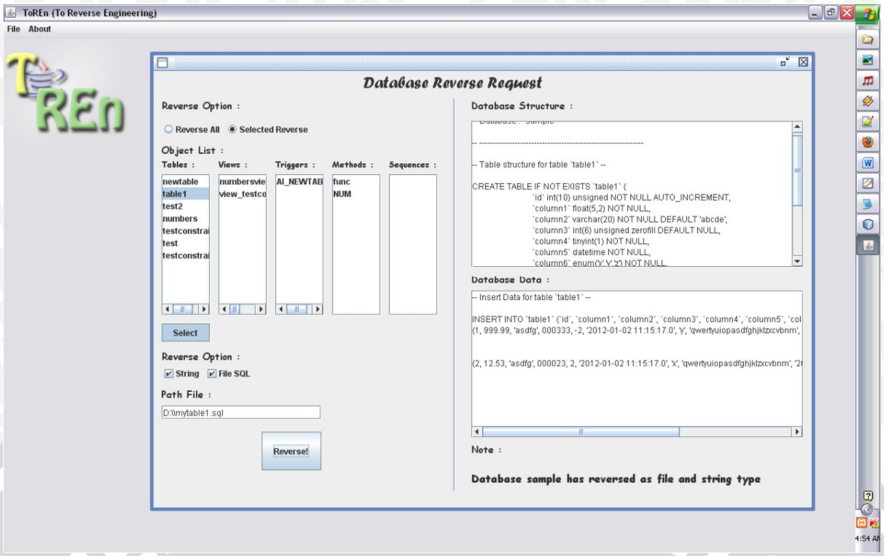
Pengujian ini bertujuan untuk mengetahui apakah ToREn API dapat bekerja sesuai fungsionalitasnya dalam DBMS MySQL. Pengujian pada DBMS MySQL dibagi menjadi 5 macam pengujian. Pengujian *point* (f.) tidak dilakukan pada DBMS MySQL karena pada DBMS ini untuk melakukan *auto numbering* tidak dengan menggunakan *sequence function*.

#### 5.1.1.1 Pengujian *Reverse Engineering Database Point* (a.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah tabel beserta komponen penyusunnya. Komponen penyusun yang dimaksud dalam pengujian ini antara lain :

- Kolom yang memiliki beberapa macam tipe data
- *Null constraint*
- *Default value*
- *Index*
- *Primary key*
- *Unique key*
- Data di dalam tabel

Gambar 5.1 menunjukkan proses *reverse engineering database* untuk point (a.) :




**Gambar 5.1** *Reverse Engineering Database Point* (a.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextField` aplikasi
2. Berupa *file* mytable1.sql

Tabel 5.1 berisi SQL *script* hasil r*everse engineering* :

**Table 5.1** SQL Script Hasil Reverse Engineering Database Point (a.)

| No. | Nama Tabel | *SQL Script* |
|---|---|---|
| 1. | **TABLE1** | `-- Table structure for table `table1` --`<br><br>`CREATE TABLE IF NOT EXISTS `table1` (`<br>`  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,`<br>`  `column1` float(5,2) NOT NULL,`<br>`  `column2` varchar(20) NOT NULL DEFAULT 'abcde',`<br>`  `column3` int(6) unsigned zerofill DEFAULT NULL,`<br>`  `column4` tinyint(1) NOT NULL,`<br>`  `column5` datetime NOT NULL,`<br>`  `column6` enum('x','y','z') NOT NULL,`<br>`  `column7` text NOT NULL,`<br>`  `column8` date NOT NULL,`<br>`  `column9` tinyint(1) NOT NULL,`<br>`  PRIMARY KEY (`id`),`<br>`  UNIQUE INDEX `column1` (`column1` ASC, `column8` ASC),`<br>`  INDEX `column3` (`column3`)`<br>`) ENGINE = InnoDB DEFAULT CHARSET = latin1`<br>`AUTO_INCREMENT = 3;`<br><br>`-- Insert Data for table `table1` --`<br><br>`INSERT INTO `table1` (`id`, `column1`, `column2`, `column3`, `column4`, `column5`, `column6`, `column7`, `column8`, `column9`) VALUES` |

| | | |
|---|---|---|
| | | (1, 999.99, 'asdfg', 000333, -2, '2012-01-02 11:15:17.0', 'y', 'qwertyuiopasdfghjklzxcvbnm', '2012-01-06', 0),<br><br>(2, 12.53, 'asdfg', 000023, 2, '2012-01-02 11:15:17.0', 'x', 'qwertyuiopasdfghjklzxcvbnm', '2012-01-06', 1); |

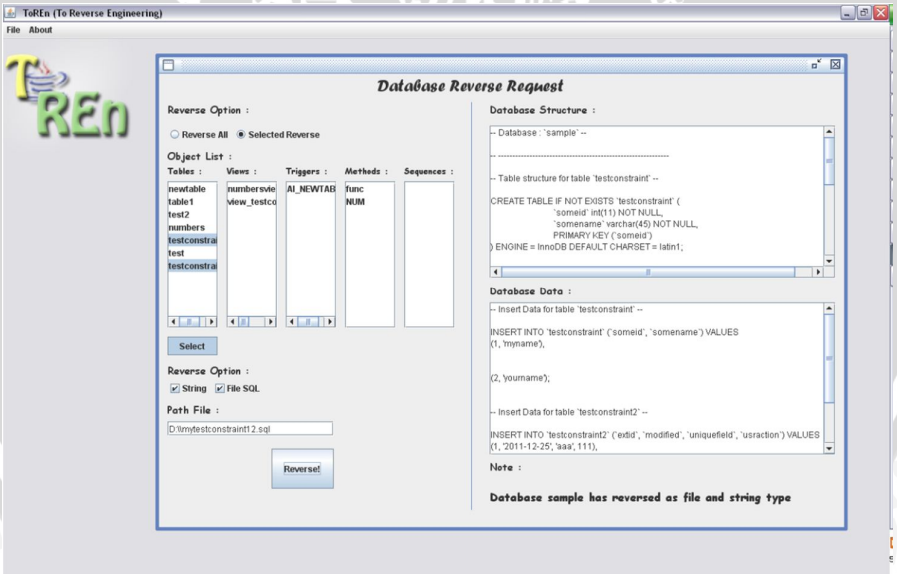
**Sumber :** Pengujian

### 5.1.1.2 Pengujian *Reverse Engineering Database Point* (b.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur dua buah tabel berelasi beserta komponen penyusunnya. Komponen penyusun yang dimaksud dalam pengujian ini antara lain :

- Kolom yang memiliki beberapa macam tipe data
- *Primary key*
- *Foreign key*
- Data di dalam tabel

Gambar 5.2 menunjukkan proses *reverse engineering database* untuk *point* (b.) :




**Gambar 5.2** *Reverse Engineering Database Point* (a.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextField` aplikasi

2. Berupa *file* mytestconstraint12.sql

Tabel 5.2 berisi SQL *script* hasil *reverse engineering :*

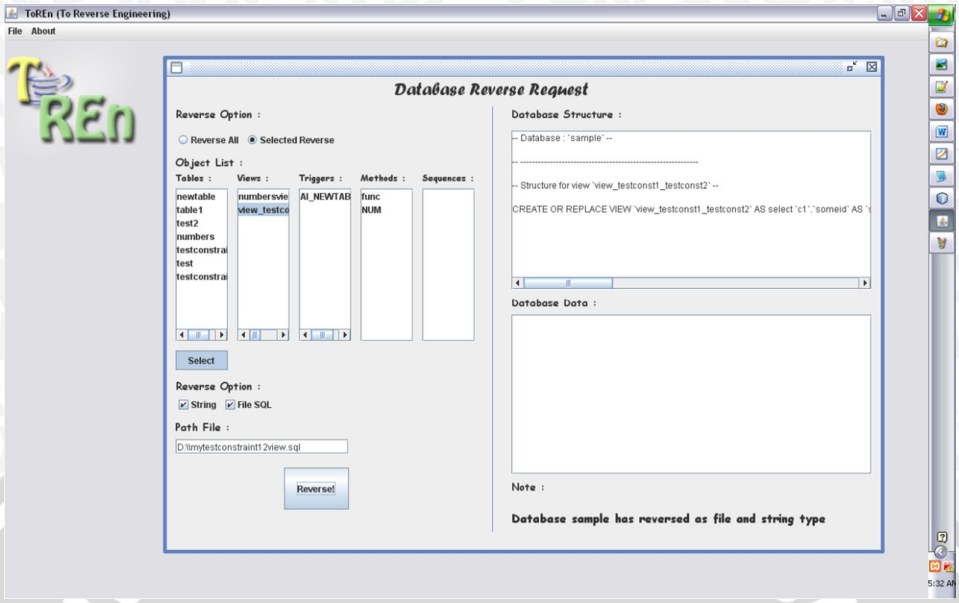
**Table 5.2** SQL Script Hasil Reverse Engineering Database Point (b.)

| No. | Nama Tabel | *SQL Script* |
|---|---|---|
| 1. | TESTCONSTRAINTS | `-- Table structure for table `testconstraint` --`<br><br>`CREATE TABLE IF NOT EXISTS `testconstraint` (`<br>`  `someid` int(11) NOT NULL,`<br>`  `somename` varchar(45) NOT NULL,`<br>`  PRIMARY KEY (`someid`)`<br>`) ENGINE = InnoDB DEFAULT CHARSET = latin1;`<br><br>`-- Insert Data for table `testconstraint` --`<br><br>`INSERT INTO `testconstraint` (`someid`,`<br>`somename`) VALUES`<br>`(1, 'myname'),`<br><br>`(2, 'yourname');` |
| 2. | TESTCONSTRAINTS2 | `-- Table structure for table `testconstraint2` --`<br><br>`CREATE TABLE IF NOT EXISTS `testconstraint2` (`<br>`  `extid` int(11) NOT NULL,`<br>`  `modified` date DEFAULT NULL,`<br>`  `uniquefield` varchar(45) NOT NULL,`<br>`  `usraction` int(11) NOT NULL,`<br>`  UNIQUE INDEX `uniquefld_idx``<br>`(`uniquefield` ASC),`<br>`  UNIQUE INDEX `unique_2_fields_idx``<br>`(`modified` ASC, `uniquefield` ASC),`<br>`  INDEX `testconstraints_id_fk` (`extid`),`<br>`  CONSTRAINT `testconstraints_id_fk``<br>`  FOREIGN KEY (`extid`)`<br>`  REFERENCES `testconstraint`<br>`(`someid`)`<br>`  ON DELETE NO ACTION`<br>`  ON UPDATE NO ACTION`<br>`) ENGINE = InnoDB DEFAULT CHARSET = latin1;`<br><br>`-- Insert Data for table `testconstraint2` --`<br><br>`INSERT INTO `testconstraint2` (`extid`,`<br>`modified`, `uniquefield`, `usraction`) VALUES`<br>`(1, '2011-12-25', 'aaa', 111),`<br><br>`(2, '2011-12-28', 'bbb', 222);` |

**Sumber :** Pengujian

### 5.1.1.3 Pengujian *Reverse Engineering Database Point* (c.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah *view*. Gambar 5.3 menunjukkan proses *reverse engineering database* untuk *point* (c.) :




**Gambar 5.3** *Reverse Engineering Database Point* (c.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextField` aplikasi
2. Berupa *file* mytestconstraint12view.sql

Tabel 5.3 berisi SQL *script* hasil *reverse engineering :*

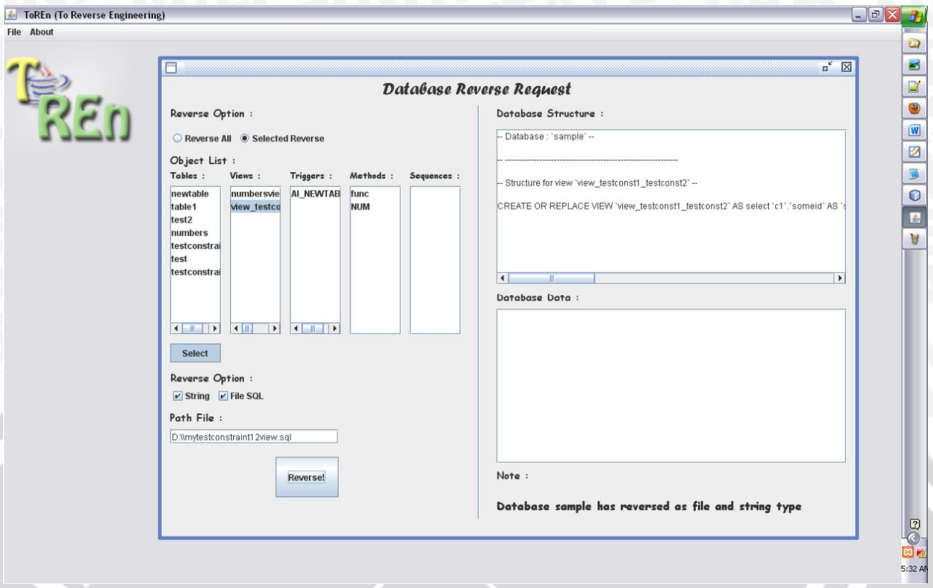
**Table 5.3** SQL Script Hasil Reverse Engineering Database Point (c.)

| No. | Nama View | *SQL Script* |
|---|---|---|
| 1. | VIEW_TESTCONST1_TESTCONST2 | `-- Structure for view` `` `view_testconst1_testconst2` `` `--` `CREATE OR REPLACE VIEW` `` `view_testconst1_testconst2` `` `AS select` `` `c1`.`someid` `` `AS` `` `someid`,`c1`.`somename` `` `AS` `` `somename`,`c2`.`modified` `` `AS` `` `modified`,`c2`.`uniquefield` `` `AS` `` `uniquefield`,`c2`.`usraction` `` `AS` `usraction` `from` `` `sample`.`testconstraint` `` `c1` `join` `` `sample`.`testconstraint2` `` `c2` `where` `` (`c1`.`someid` = `c2`.`extid`); `` |

**Sumber :** Pengujian

### 5.1.1.4 Pengujian *Reverse Engineering Database Point* (d.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah *trigger*. Gambar 5.4 menunjukkan proses *reverse engineering database* untuk *point* (d.) :




**Gambar 5.4** *Reverse Engineering Database Point* (d.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextField` aplikasi
2. Berupa *file* mytrigger.sql

Tabel 5.4 berisi SQL *script* hasil r*everse engineering :*

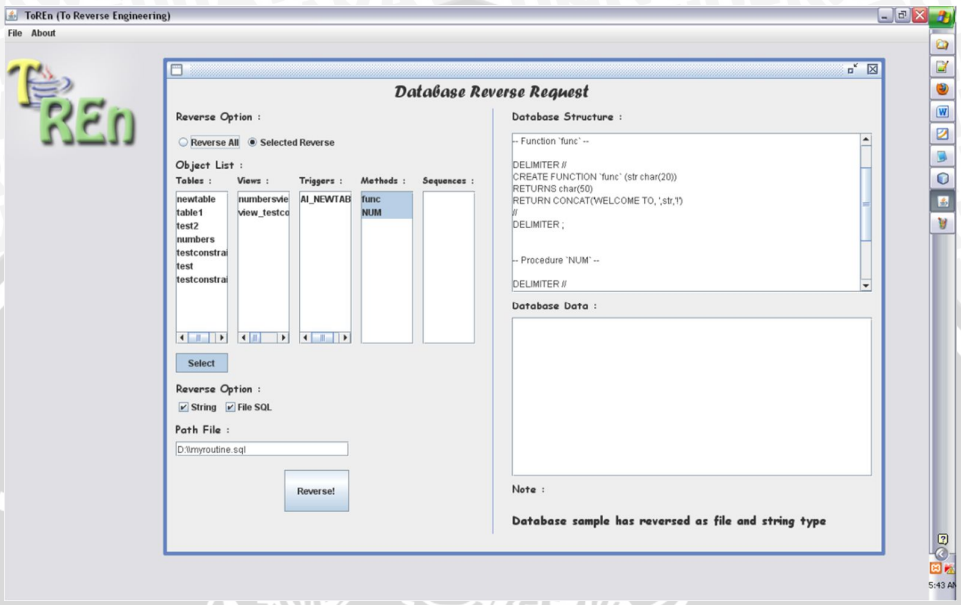
**Table 5.4** SQL Script Hasil Reverse Engineering Database Point (d.)

| No. | Nama Trigger | *SQL Script* |
|---|---|---|
| 1. | AI_NEWTABLE | `-- Trigger `AI_NEWTABLE` --`<br><br>`DROP TRIGGER IF EXISTS `AI_NEWTABLE`;`<br>`DELIMITER //`<br>`CREATE TRIGGER `AI_NEWTABLE` AFTER INSERT ON `newtable``<br>`FOR EACH ROW`<br>`BEGIN`<br>`INSERT INTO TEST (test_id, test_name, test_date)`<br>`VALUES (NEW.id, NEW.somename, NEW.somedate);`<br>`END;`<br>`//`<br>`DELIMITER ;` |

**Sumber :** Pengujian

### 5.1.1.5 Pengujian *Reverse Engineering Database Point* (e.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah *routine*. Gambar 5.5 menunjukkan proses *reverse engineering database* untuk *point* (e.) :




**Gambar 5.5** *Reverse Engineering Database Point* (e.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextField` aplikasi
2. Berupa *file* myroutine.sql

Tabel 5.5 berisi SQL *script* hasil *reverse engineering :*

**Table 5.5** SQL Script Hasil Reverse Engineering Database Point (e.)

| No. | Nama Routine | *SQL Script* |
|---|---|---|
| 1. | FUNC | `-- Function `func` --`<br><br>`DELIMITER //`<br>`CREATE FUNCTION func (str char(20))`<br>`RETURNS char(50)`<br>`RETURN CONCAT('WELCOME TO, ',str,'!')`<br><br>`//`<br>`DELIMITER ;` |
| 2. | NUM | `-- Procedure `NUM` --`<br><br>`DELIMITER //`<br>`CREATE PROCEDURE NUM (IN n int, OUT e varchar(100), OUT f varchar(100))`<br>`SELECT num,en, fr INTO n,e,f FROM NUMBERS WHERE NUM=n` |

| | | |
|---|---|---|
| | | //<br>DELIMITER ; |

**Sumber :** Pengujian

### 5.1.2 Pengujian *Reverse Engineering Database* menjadi SQL *Script* DBMS Oracle

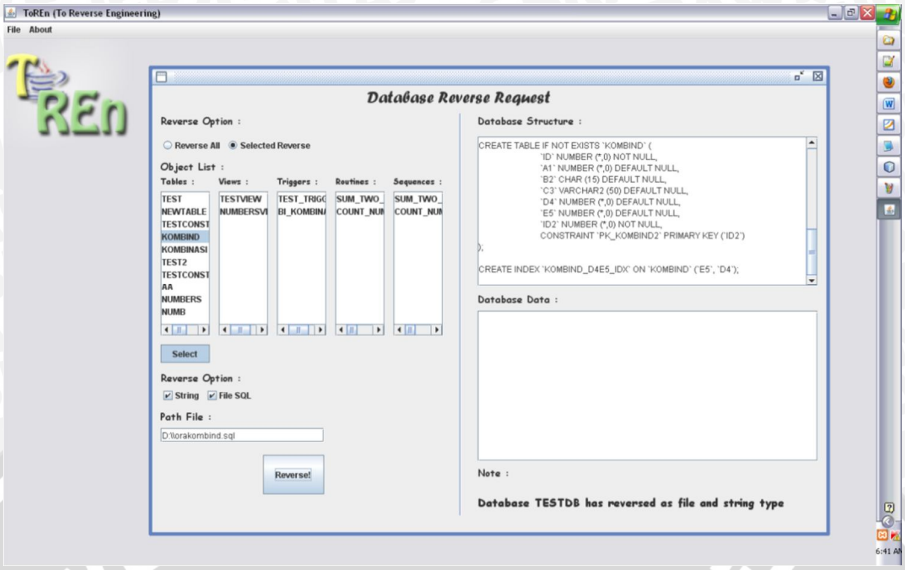
Pengujian ini bertujuan untuk mengetahui apakah ToREn API dapat bekerja sesuai fungsionalitasnya dalam DBMS Oracle. Pengujian pada DBMS Oracle dibagi menjadi 6 macam pengujian.

#### 5.1.2.1 Pengujian *Reverse Engineering Database Point* (a.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah tabel beserta komponen penyusunnya. Komponen penyusun yang dimaksud dalam pengujian ini antara lain :

- Kolom yang memiliki beberapa macam tipe data
- *Null constraint*
- *Default value*
- *Index*
- *Primary key*
- *Unique key*
- Data di dalam tabel

Gambar 5.6 menunjukkan proses *reverse engineering database* point (a.) :




**Gambar 5.6** *Reverse Engineering Database Point* (a.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextFeild` aplikasi
2. Berupa *file* orakombind.sql

Tabel 5.6 berisi SQL *script* hasil r*everse engineering :*

**Table 5.6** SQL Script Hasil Reverse Engineering Database Point (a.)

| No. | Nama Tabel | *SQL Script* |
|---|---|---|
| 1. | **KOMBIND** | `-- Table structure for table `KOMBIND` --`<br><br>`CREATE TABLE IF NOT EXISTS KOMBIND (`<br>`ID NUMBER (*,0) NOT NULL,`<br>`A1 NUMBER (*,0) DEFAULT NULL,`<br>`B2 CHAR (15) DEFAULT NULL,`<br>`C3 VARCHAR2 (50) DEFAULT NULL,`<br>`D4 NUMBER (*,0) DEFAULT NULL,`<br>`E5 NUMBER (*,0) DEFAULT NULL,`<br>`ID2 NUMBER (*,0) NOT NULL,`<br>`CONSTRAINT PK_KOMBIND2 PRIMARY KEY (ID2));`<br><br>`CREATE INDEX KOMBIND_D4E5_IDX ON KOMBIND (E5, D4);` |

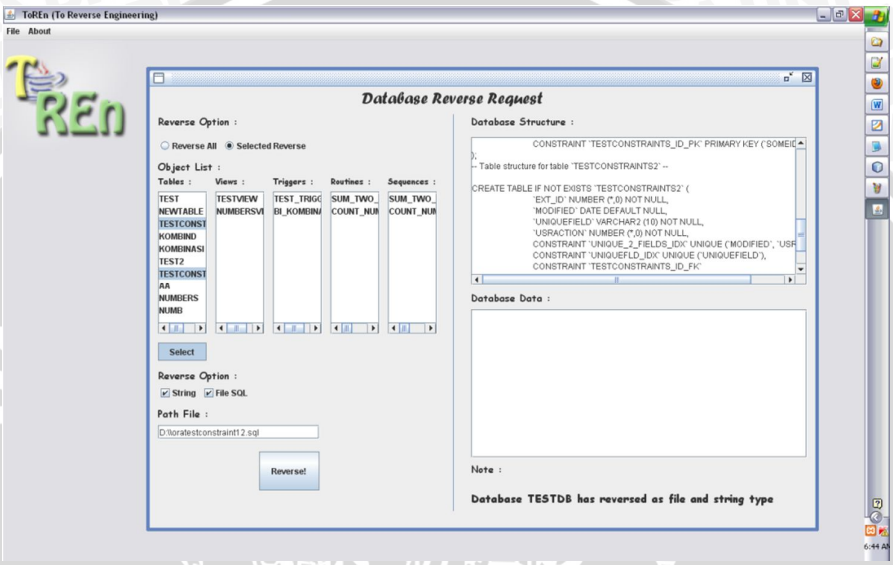
**Sumber :** Pengujian

### 5.1.2.2 Pengujian *Reverse Engineering Database Point* (b.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur dua buah tabel berelasi beserta komponen penyusunnya. Komponen penyusun yang dimaksud dalam pengujian ini antara lain :

- Kolom yang memiliki beberapa macam tipe data
- *Primary key*
- *Foreign key*
- Data di dalam tabel

Gambar 5.7 menunjukkan proses *reverse engineering database point* (b.) :




**Gambar 5.7** *Reverse Engineering Database Point* (a.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

3. Berupa String yang tampil pada `jTextField` aplikasi
4. Berupa *file* oratestconstraint12.sql

Tabel 5.7 berisi SQL *script* hasil r*everse engineering :*

**Table 5.7** SQL Script Hasil Reverse Engineering Database Point (b.)

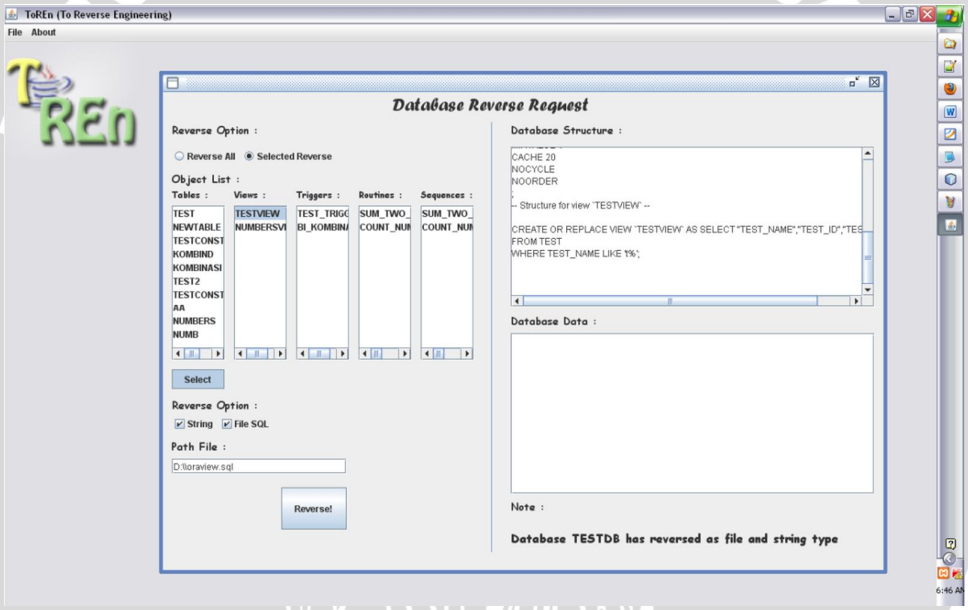
| No. | Nama Tabel | *SQL Script* |
|---|---|---|
| **1.** | **TESTCONSTRAINTS** | `-- Table structure for table `TESTCONSTRAINTS` --`<br><br>`CREATE TABLE IF NOT EXISTS TESTCONSTRAINTS (`<br>`SOMEID NUMBER (*,0) NOT NULL,`<br>`SOMENAME VARCHAR2 (10) NOT NULL,`<br>`CONSTRAINT TESTCONSTRAINTS_ID_PK PRIMARY`<br>`KEY (SOMEID)`<br>`);` |
| **2.** | **TESTCONSTRAINTS2** | `--Table structure for table `TESTCONSTRAINTS2`--`<br>`CREATE TABLE IF NOT EXISTS TESTCONSTRAINTS2 (`<br>`EXT_ID NUMBER (*,0) NOT NULL,`<br>`MODIFIED DATE DEFAULT NULL,`<br>`UNIQUEFIELD VARCHAR2 (10) NOT NULL,`<br>`USRACTION NUMBER (*,0) NOT NULL,`<br>`CONSTRAINT UNIQUE_2_FIELDS_IDX UNIQUE` |

| | | |
|---|---|---|
| | | (MODIFIED, USRACTION),<br>CONSTRAINT UNIQUEFLD_IDX UNIQUE<br>(UNIQUEFIELD),<br>CONSTRAINT TESTCONSTRAINTS_ID_FK<br>FOREIGN KEY (EXT_ID)<br>REFERENCES TESTCONSTRAINTS(SOMEID)<br>ON DELETE CASCADE<br>); |

**Sumber :** Pengujian

### 5.1.2.3 Pengujian *Reverse Engineering Database Point* (c.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah *view*. Gambar 5.8 menunjukkan proses *reverse engineering database* untuk *point* (c.) :




**Gambar 5.8** *Reverse Engineering Database Point* (c.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextField` aplikasi
2. Berupa *file* oraview.sql

Tabel 5.8 berisi SQL *script* hasil r*everse engineering :*

**Table 5.8** SQL Script Hasil Reverse Engineering Database Point (c.)

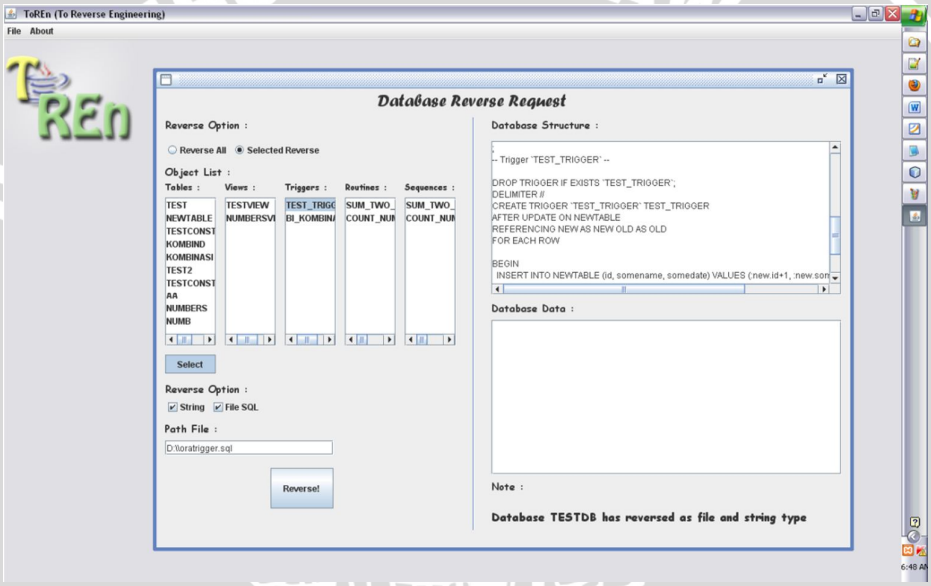
| No. | Nama View | *SQL Script* |
|---|---|---|
| 1. | TEST_VIEW | `-- Structure for view `TESTVIEW` --` |

| | | |
|---|---|---|
| | | CREATE OR REPLACE VIEW TESTVIEW AS SELECT<br>"TEST_NAME","TEST_ID","TEST_DATE"<br>FROM TEST<br>WHERE TEST_NAME LIKE 't%'; |

**Sumber :** Pengujian

### 5.1.2.4 Pengujian *Reverse Engineering Database Point* (d.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah *trigger*. Gambar 5.9 menunjukkan proses *reverse engineering database* untuk *point* (d.) :




**Gambar 5.9** *Reverse Engineering Database Point* (d.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextField` aplikasi
2. Berupa *file* mytrigger.sql

Tabel 5.9 berisi SQL *script* hasil r*everse engineering :*

**Table 5.9** SQL Script Hasil Reverse Engineering Database Point (d.) :

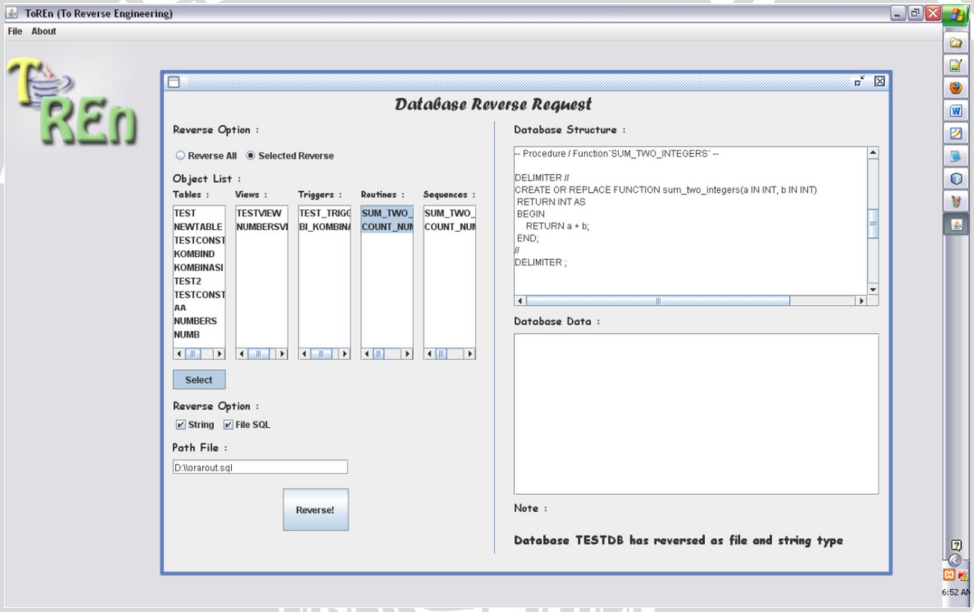
| No. | Nama Trigger | *SQL Script* |
|---|---|---|
| 1. | TEST_TRIGGER | -- Trigger `TEST_TRIGGER` --<br>DROP TRIGGER IF EXISTS TEST_TRIGGER;<br>DELIMITER //<br>CREATE TRIGGER TEST_TRIGGER TEST_TRIGGER<br>AFTER UPDATE ON NEWTABLE |

| | | |
|---|---|---|
| | | REFERENCING NEW AS NEW OLD AS OLD<br>FOR EACH ROW<br>BEGIN<br>  INSERT INTO NEWTABLE (id, somename, somedate)<br>VALUES (:new.id+1, :new.somename, :new.somedate);<br>END TEST_TRIGGER;<br>//<br>DELIMITER ; |

**Sumber :** Pengujian

#### 5.1.2.5 Pengujian *Reverse Engineering Database Point* (e.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah *routine*. Gambar 5.10 menunjukkan proses *reverse engineering database* untuk *point* (e.) :




**Gambar 5.10** *Reverse Engineering Database Point* (e.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextField` aplikasi
2. Berupa *file* myroutine.sql

Tabel 5.10 berisi SQL *script* hasil r*everse engineering :*

**Table 5.10** SQL Script Hasil Reverse Engineering Database Point (e.)

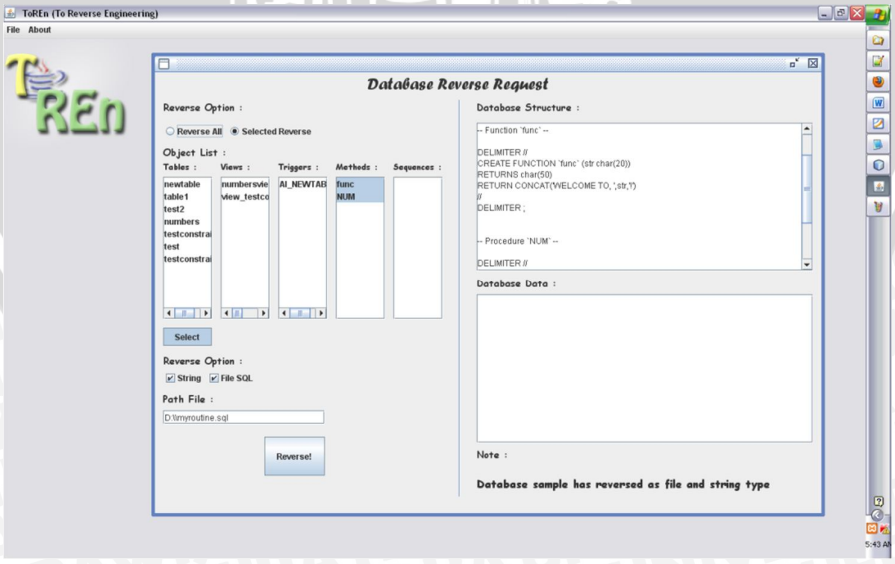
| No. | Nama Routine | *SQL Script* |
|---|---|---|
| 1. | SUM_TWO_INTEGERS | -- Procedure / Function`SUM_TWO_INTEGERS`<br>--<br>DELIMITER // |

| | | |
|---|---|---|
| | | CREATE OR REPLACE FUNCTION<br>sum_two_integers(a IN INT, b IN INT)<br>RETURN INT AS<br>BEGIN<br>RETURN a + b;<br>END;<br>//<br>DELIMITER ; |
| 2. | COUNT_NUMBERS_BY_LANG | -- Procedure /<br>Function`COUNT_NUMBERS_BY_LANG` --<br>DELIMITER //<br>CREATE OR REPLACE PROCEDURE<br>count_numbers_by_lang(var_lang in<br>varchar2, var_num out integer)<br>IS<br>var_temp_n INTEGER;<br>BEGIN<br>SELECT COUNT(DISTINCT var_lang)<br>INTO var_temp_n<br>FROM NUMBERS<br>WHERE var_lang IS NOT NULL;<br>var_num := var_temp_n;<br>return;<br>END;<br>//<br>DELIMITER ; |

**Sumber :** Pengujian

### 5.1.2.6 Pengujian *Reverse Engineering Database Point* (f.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah *trigger*. Gambar 5.11 menunjukkan proses *reverse engineering database* untuk *point* (f.) :




**Gambar 5.11** *Reverse Engineering Database Point* (f.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextField` aplikasi
2. Berupa *file* mysequence.sql

Tabel 5.11 berisi SQL *script* hasil r*everse engineering :*

**Table 5.11** SQL Script Hasil Reverse Engineering Database Point (f.)

| No. | Nama Sequence | *SQL Script* |
|---|---|---|
| 1. | NEWTABLE_SEQ | CREATE SEQUENCE NEWTABLE_SEQ<br>INCREMENT BY 1<br>START WITH 1<br>MAXVALUE 999999999999999999999999999<br>MINVALUE 1<br>CACHE 0<br>NOCYCLE<br>NOORDER |

**Sumber :** Pengujian

### 5.1.3 Pengujian *Reverse Engineering Database* menjadi SQL *Script* DBMS PostgreSQL

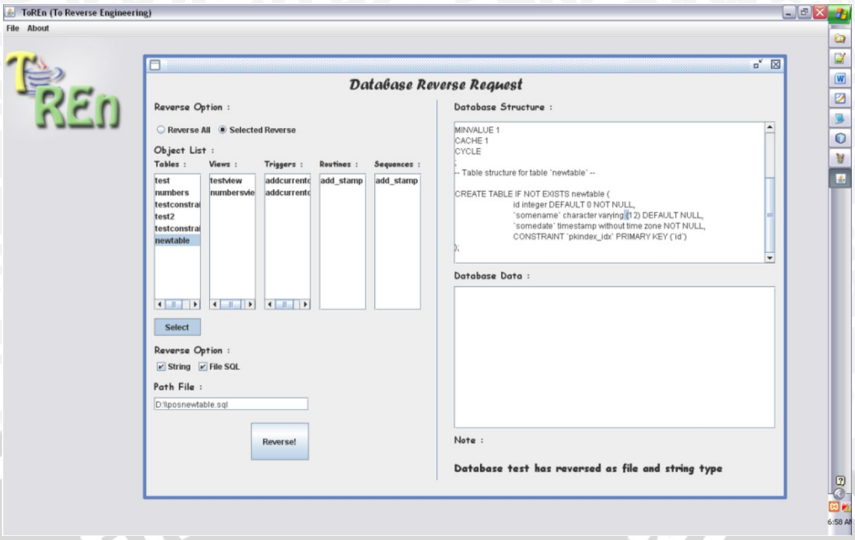
Pengujian ini bertujuan untuk mengetahui apakah ToREn API dapat bekerja sesuai fungsionalitasnya dalam DBMS PostgreSQL. Pengujian pada DBMS PostgreSQL dibagi menjadi 6 macam pengujian.

#### 5.1.3.1 Pengujian *Reverse Engineering Database Point* (a.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah tabel beserta komponen penyusunnya. Komponen penyusun yang dimaksud dalam pengujian ini antara lain :

- Kolom yang memiliki beberapa macam tipe data
- *Null constraint*
- *Default value*
- *Index*
- *Primary key*
- *Unique key*
- Data di dalam tabel

Gambar 5.12 menunjukkan proses *reverse engineering database* point (a.) :




**Gambar 5.12** *Reverse Engineering Database Point* (a.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

3. Berupa String yang tampil pada `jTextField` aplikasi
4. Berupa *file* posnewtable.sql

Tabel 5.12 berisi SQL *script* hasil *Reverse Engineering Database :*

**Table 5.12** SQL Script Hasil Reverse Engineering Database Point (a.)

| No. | Nama Tabel | *SQL Script* |
|---|---|---|
| **1.** | **newtable** | `-- Table structure for table `newtable` --`<br><br>`CREATE TABLE IF NOT EXISTS newtable (`<br>`id integer DEFAULT 0 NOT NULL,`<br>`somename character varying (12) DEFAULT NULL,`<br>`somedate timestamp without time zone NOT NULL,`<br>`CONSTRAINT pkindex_idx PRIMARY KEY (id)`<br>`);` |

**Sumber :** Pengujian

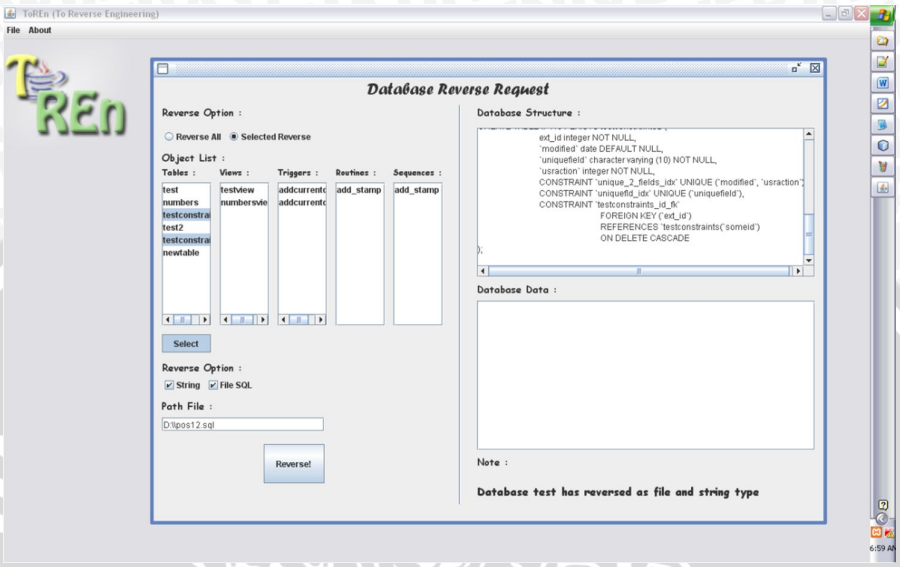
### 5.1.3.2 Pengujian *Reverse Engineering Database Point* (b.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur dua buah tabel berelasi beserta komponen penyusunnya. Komponen penyusun yang dimaksud dalam pengujian ini antara lain :

- Kolom yang memiliki beberapa macam tipe data
- *Primary key*
- *Foreign key*

- Data di dalam tabel

Gambar 5.13 menunjukkan proses *reverse engineering database* untuk *point* (b.) :




**Gambar 5.13** *Reverse Engineering Database Point* (a.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextField` aplikasi
2. Berupa *file* postestconstraint12.sql

Tabel 5.13 berisi SQL *script* hasil r*everse engineering :*

**Table 5.13** SQL Script Hasil Reverse Engineering Database Point (b.)

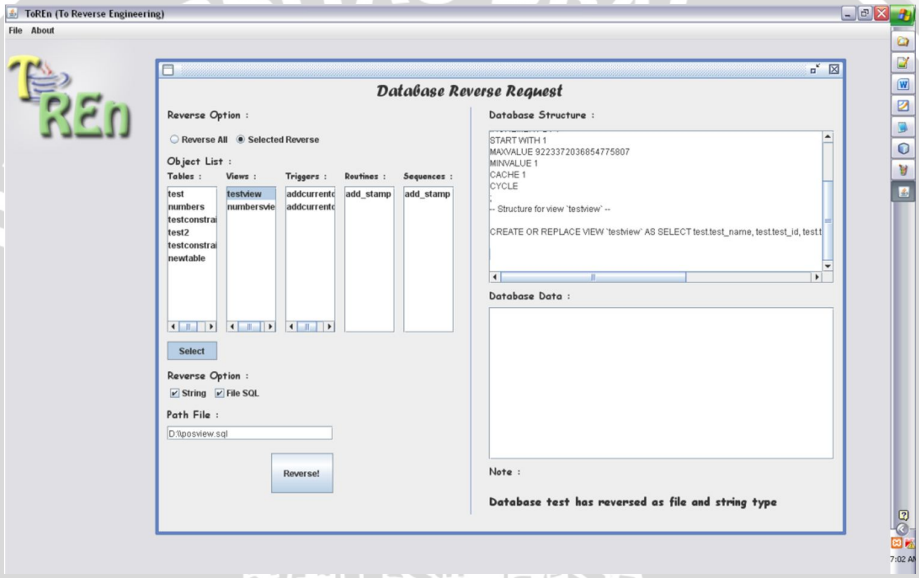
| No. | Nama Tabel | *SQL Script* |
|---|---|---|
| **1.** | **testconstraints** | `-- Table structure for table `testconstraints` --`<br>`CREATE TABLE IF NOT EXISTS testconstraints (`<br>`someid integer NOT NULL,`<br>`somename character varying (10) NOT NULL,`<br>`CONSTRAINT testconstraints_id_pk PRIMARY KEY (someid));` |
| **2.** | **testconstraints2** | `-- Table structure for table `testconstraints2` --`<br>`CREATE TABLE IF NOT EXISTS testconstraints2 (`<br>`ext_id integer NOT NULL,`<br>`modified date DEFAULT NULL,`<br>`uniquefield character varying (10) NOT NULL,`<br>`usraction integer NOT NULL,`<br>`CONSTRAINT unique_2_fields_idx UNIQUE (modified, usraction),`<br>`CONSTRAINT `uniquefld_idx` UNIQUE (uniquefield),`<br>`CONSTRAINT `testconstraints_id_fk`` |

| | | |
|---|---|---|
| | | FOREIGN KEY (`ext_id`)<br>REFERENCES `testconstraints(`someid`)<br>ON DELETE CASCADE<br>); |

**Sumber :** Pengujian

### 5.1.3.3 Pengujian *Reverse Engineering Database Point* (c.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah *view*. Gambar 5.14 menunjukkan proses *reverse engineering database* untuk *point* (c.) :




**Gambar 5.14** *Reverse Engineering Database Point* (c.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextField` aplikasi
2. Berupa *file* posview.sql

Tabel 5.14 berisi SQL *script* hasil *Reverse Engineering Database :*

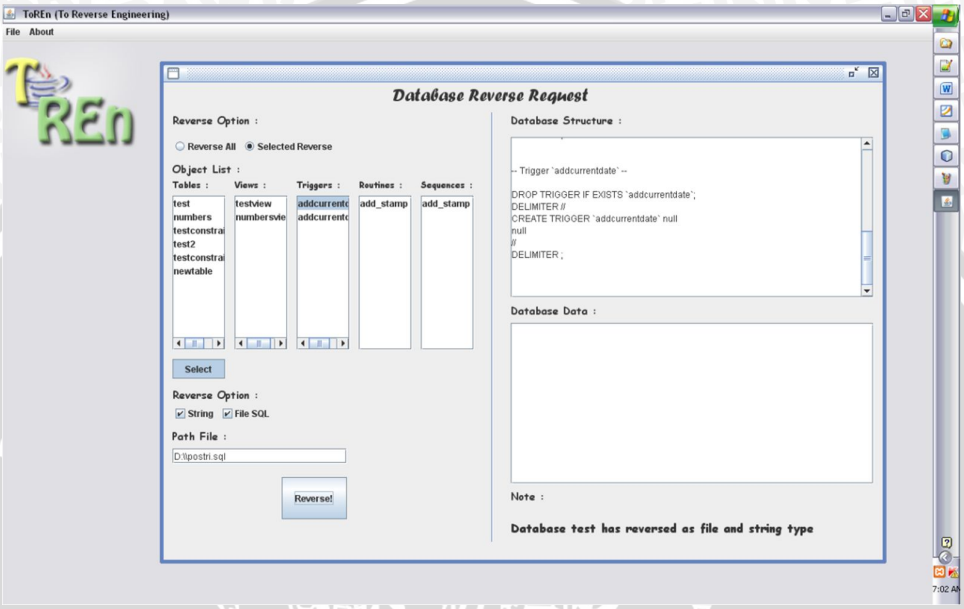
**Table 5.14** SQL Script Hasil Reverse Engineering Database Point (c.)

| No. | Nama View | *SQL Script* |
|---|---|---|
| 1. | view_testview | -- Structure for view `testview` --<br><br>CREATE OR REPLACE VIEW testview AS SELECT test.test_name, test.test_id, test.test_date FROM test WHERE (test.test_name ~~ 't%'::text); |

**Sumber :** Pengujian

### 5.1.3.4 Pengujian *Reverse Engineering Database Point* (d.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah *trigger*. Gambar 5.15 menunjukkan proses *reverse engineering database* untuk *point* (d.) :




**Gambar 5.15** *Reverse Engineering Database Point* (d.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextField` aplikasi
2. Berupa *file* postrigger.sql

Tabel 5.15 berisi SQL *script* hasil *reverse engineering :*

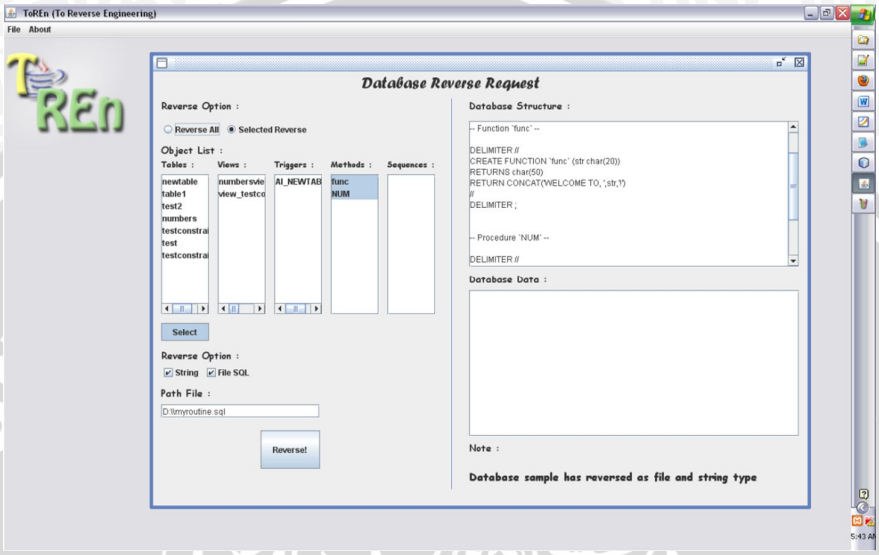
**Table 5.15** SQL Script Hasil Reverse Engineering Database Point (d.)

| No. | Nama Trigger | *SQL Script* |
|---|---|---|
| 1. | addcurrentdate | `-- Trigger `addcurrentdate` --`<br><br>`DROP TRIGGER IF EXISTS addcurrentdate;`<br>`DELIMITER //`<br>`CREATE TRIGGER addcurrentdate`<br>`BEFORE INSERT OR UPDATE`<br>`ON newtable FOR EACH ROW`<br>`EXECUTE PROCEDURE add_stamp();`<br>`//`<br>`DELIMITER ;` |

**Sumber :** Pengujian

### 5.1.3.5 Pengujian *Reverse Engineering Database Point* (e.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah *routine*. Gambar 5.16 menunjukkan proses *reverse engineering database* untuk *point* (e.) :




**Gambar 5.16** *Reverse Engineering Database Point* (e.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

1. Berupa String yang tampil pada `jTextField` aplikasi
2. Berupa *file* posroutine.sql

Tabel 5.16 berisi SQL *script* hasil r*everse engineering :*

**Table 5.16** SQL Script Hasil Reverse Engineering Database Point (e.)

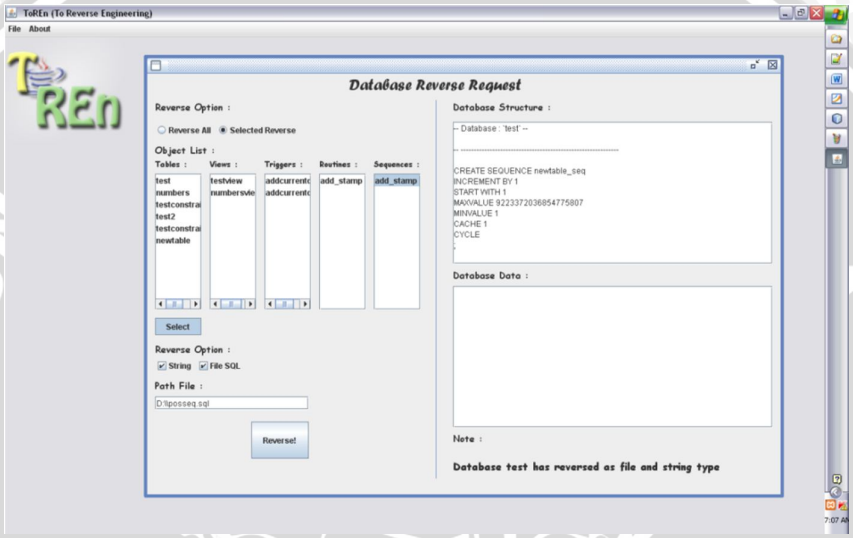
| No. | Nama Routine | *SQL Script* |
|---|---|---|
| 1. | func | `-- Function `func` --`<br><br>`DELIMITER //`<br><br>`CREATE FUNCTION func (str char(20))`<br>`RETURNS char(50)`<br>`RETURN CONCAT('WELCOME TO, ',str,'!')`<br>`//`<br>`DELIMITER ;` |
| 2. | NUM | `-- Procedure `NUM` --`<br><br>`DELIMITER //`<br><br>`CREATE PROCEDURE NUM (IN n int, OUT e varchar(100), OUT f varchar(100))` |

| | | |
|---|---|---|
| | | SELECT num,en, fr INTO n,e,f FROM NUMBERS WHERE NUM=n<br>//<br>DELIMITER ; |

**Sumber :** Pengujian

### 5.1.3.6 Pengujian *Reverse Engineering Database Point* (f.)

Pengujian ini bertujuan untuk mengetahui bahwa ToREn API dapat me-*reverse engineering* struktur sebuah *trigger*. Gambar 5.17 menunjukkan proses *reverse engineering database* untuk *point* (f.) :




**Gambar 5.17** *Reverse Engineering Database Point* (f.) pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Pada pengujian ini, aplikasi menghasilkan dua buah keluaran yaitu :

3. Berupa String yang tampil pada `jTextField` aplikasi
4. Berupa *file* possequence.sql

Tabel 5.17 berisi SQL *script* hasil r*everse engineering :*

**Table 5.17** SQL Script Hasil Reverse Engineering Database Point (f.)

| No. | Nama Sequence | *SQL Script* |
|---|---|---|
| 1. | newtable_seq | CREATE SEQUENCE newtable_seq<br>INCREMENT BY 1<br>START WITH 1<br>MAXVALUE 9223372036854775807<br>MINVALUE 1<br>CACHE 1<br>CYCLE; |

**Sumber :** Pengujian

## 5.2 Pengujian Eksekusi SQL *Script*

Pengujian ini bertujuan untuk mengetahui apakah SQL *script* hasil dari *reverse engineering database* menggunakan ToREn API dapat dieksekusi pada DBMS asalnya.

Pengujian dibagi menjadi tiga bagian, yang masing – masing bagian mewakili satu buah DBMS.

### 5.2.1 Pengujian Eksekusi SQL *Script* DBMS MySQL

Pengujian ini bertujuan untuk mengetahui apakah SQL *script* hasil dari *reverse engineering database* menggunakan ToREn API dapat dieksekusi pada DBMS MySQL. Tabel 5.18 menunjukkan hasil pengujian SQL *script* DBMS MySQL :

**Table 5.18** Hasil pengujian SQL *script* pada DBMS MySQL

| No. | Nama Objek | Hasil Pengujian | Kesesuaian Struktur |
|---|---|---|---|
| 1 | Objek *Point* (a.) | bisa dieksekusi | sesuai |
| 2 | Objek *Point* (b.) | bisa dieksekusi | sesuai |
| 3 | Objek *Point* (c.) | bisa dieksekusi | sesuai |
| 4 | Objek *Point* (d.) | bisa dieksekusi | sesuai |
| 5 | Objek *Point* (e.) | bisa dieksekusi | sesuai |

**Sumber :** Pengujian

Hasil pengujian pada Tabel 5.18 dapat disimpulkan bahwa SQL *script* dapat di eksekusi pada DBMS MySQL.

### 5.2.2 Pengujian Eksekusi SQL *Script* DBMS Oracle

Pengujian ini bertujuan untuk mengetahui apakah SQL *script* hasil dari *reverse engineering database* menggunakan ToREn API dapat dieksekusi pada DBMS Oracle. Tabel 5.19 menunjukkan hasil pengujian SQL *script* DBMS Oracle :

**Table 5.19** Hasil pengujian SQL *script* pada DBMS Oracle

| No. | Nama Objek | Hasil Pengujian | Kesesuaian Struktur |
|---|---|---|---|
| 1 | Objek *Point* (a.) | bisa dieksekusi | sesuai |
| 2 | Objek *Point* (b.) | bisa dieksekusi | sesuai |
| 3 | Objek *Point* (c.) | bisa dieksekusi | sesuai |
| 4 | Objek *Point* (d.) | bisa dieksekusi | sesuai |
| 5 | Objek *Point* (e.) | bisa dieksekusi | sesuai |

**Sumber :** Pengujian

Hasil pengujian pada Tabel 5.19 dapat disimpulkan bahwa SQL *script* dapat di eksekusi pada DBMS Oracle.

### 5.2.3 Pengujian Eksekusi SQL *Script* DBMS PosgreSQL

Pengujian ini bertujuan untuk mengetahui apakah SQL *script* hasil dari *reverse engineering database* menggunakan ToREn API dapat dieksekusi pada DBMS PostgreSQL. Tabel 5.20 menunjukkan hasil pengujian SQL *script* DBMS PostgreSQL :

**Table 5.20** Hasil pengujian SQL *script* pada DBMS PostgreSQL

| No. | Nama Objek | Hasil Pengujian | Kesesuaian Struktur |
|---|---|---|---|
| 1 | Objek *Point* (a.) | bisa dieksekusi | sesuai |
| 2 | Objek *Point* (b.) | bisa dieksekusi | sesuai |
| 3 | Objek *Point* (c.) | bisa dieksekusi | sesuai |
| 4 | Objek *Point* (d.) | bisa dieksekusi | Sesuai |
| 5 | Objek *Point* (e.) | bisa dieksekusi | Sesuai |

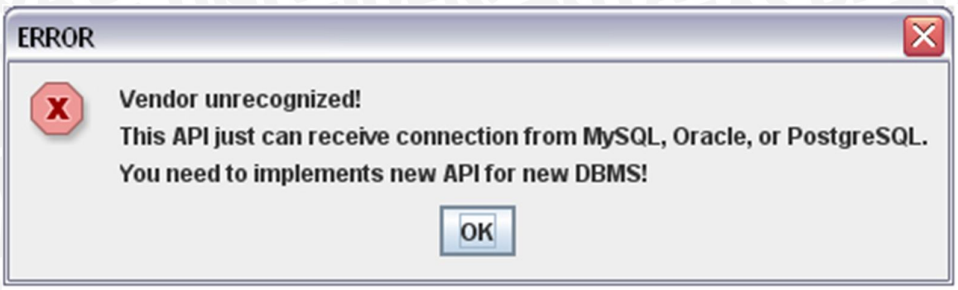
**Sumber :** Pengujian

Hasil pengujian pada Tabel 5.20 dapat disimpulkan bahwa SQL *script* dapat di eksekusi pada DBMS PostgreSQL.

## 5.3 Pengujian API terhadap Masukan Koneksi Salah

Pengujian ini bertujuan untuk mengetahui apakah ToREn API yang telah diimplementasikan dalam sebuah aplikasi *reverse engineering database,* dapat menangani apabila koneksi DBMS yang diterima salah. Dalam pengujian ini diharapkan aplikasi dapat menampilkan informasi kesalahan dikarenakan aplikasi hanya dapat menerima koneksi DBMS dari MySQL, Oracle, dan

PostgreSQL. Gambar 5.18 menunjukkan hasil pengujian ketika koneksi DBMS yang diterima tidak sesuai dengan spesifikasi ToREn API :




**Gambar 5.18** *Error Information* Akibat Kesalahan Koneksi DBMS pada Aplikasi yang Menggunakan ToREn API

**Sumber :** Pengujian

Hasil pengujian pada Gambar 5.18 dapat disimpulkan bahwa ToREn API dapat menangani apabila koneksi DBMS yang diterima salah. Penanganan dilakukan dengan pemeriksaan header koneksi yang diterima, apabila header koneksi DBMS tidak sesuai dengan syarat pemeriksaan, maka ToREn API akan melakukan *exception handling* untuk memberikan informasi kesalahan.

## 5.4 Kesimpulan Hasil Pengujian

Berdasarkan pengujian yang telah dilakukan, hasil pengujian memberikan kesimpulan bahwa secara keseluruhan sistem sudah dapat menghasilkan output yang diharapkan, yaitu aplikasi yang memanfaatkan ToREn API dapat melakukan *reverse engineering* dari tiga DBMS berbeda yaitu MySQL, Oracle, dan PostgreSQL menjadi SQL *script*. SQL *script* yang dihasilkan dari aplikasi dapat berupa output String yang ditampilkan pada *form output,* atau berupa *file* berekstensi sql. Pengujian eksekusi membuktikan bahwa SQL *script* hasil *reverse engineering* dapat dieksekusi kembali pada DBMS asal serta memiliki struktur yang sesuai dengan struktur asal. Pengujian kesalahan koneksi DBMS membuktikan bahwa ToREn API dapat melakukan *exception handling* dengan memberikan informasi apabila terjadi kesalahan koneksi DBMS.

# BAB VI
# PENUTUP

## 6.1 Kesimpulan

Berdasarkan hasil perancangan dan pengujian yang dilakukan, maka diambil kesimpulan sebagai berikut:

1. API *reverse engineering database* dirancang dengan tahap sebagai berikut :
   a. Menentukan *requirement* API *reverse engineering database*.
   b. Merancang desain API dari spesifikasi API yang dibuat berdasarkan *requirement*. Desain API merupakan kerangka dari kelas-kelas yang akan diciptakan sebagai komponen penyusun API.
   c. Menentukan fungsi-fungsi dalam setiap kelas pada API *reverse engineering database*. Perancangan fungsi-fungsi dalam setiap kelas digambarkan dalam suatu *class diagram* dengan *flow chart* sebagai diagram yang menunjukkan alur prosesnya. Secara umum, fungsi yang terdapat dalam API *reverse engineering database* terdiri dari fungsi pemeriksaan DBMS vendor, fungsi-fungsi untuk mengambil dan menyimpan metadata, fungsi melakukan *sorting database metadata component*, fungsi *generate* SQL *script*, dan fungsi pembuatan *output* berisi SQL *script*.
   d. Mengimplementasikan rancang API ke dalam bahasa pemrograman Java.
2. API mendapatkan metadata dari masing – masing DBMS (MySQL, Oracle, dan PostgreSQL) dengan cara menerima koneksi JDBC untuk DBMS yang bersangkutan. Koneksi tersebut selanjutnya digunakan untuk melakukan *querying* metadata yang dalam masing – masing DBMS disimpan pada *schema* tertentu. Selanjutnya, informasi metadata disimpan dalam objek-objek yang disediakan oleh API.
3. Penyusunan informasi metadata menjadi SQL *script* dilakukan dengan membuat kerangka SQL *script* yang berisi String SQL *script* mutlak dan variabel-variabel yang akan diubah sesuai dengan informasi metadata yang didapat dari *database* yang di-*reverse engineer*. Struktur dari kerangka SQL *script* didasarkan pada kaidah standar penulisan SQL *script*.

4. Pengujian API dilakukan dengan mengimplementasikan API dalam sebuah aplikasi untuk melakukan *reverse engineering database* multi DBMS (MySQL, Oracle, dan PostgreSQL). Hasil *reverse engineering* memiliki struktur *database* yang sama dengan *database* sumber.

## 6.2 Saran

Saran yang dapat diberikan untuk pengembangan API ini antara lain :

1. API dapat dikembangkan untuk DBMS yang lain, sehingga fungsionalitas API untuk melakukan *reverse engineering database* lebih berkembang.
2. API dapat diimplementasikan menjadi sebuah aplikasi *open source* multi DBMS (MySQL, Oracle, PostgreSQL) yang dapat melakukan *reverse engineering database*
3. API dapat dijadikan sebagai dasar pembuatan aplikasi penggambaran ER-Diagram.

## DAFTAR PUSTAKA


[ANO-11] Anonymous. 2011. *Data*. Akses dari : http://en.wikipedia.org/wiki/Data. Tanggal Akses : 31 Maret 2011.

[BLO-05] Bloch, Joshua. 2005. *How to Design a Good API and Why it Matters*. Akses dari : http://www.scribd.com/doc/33655/How-to-Design-a-Good-API-and-Why-it-Matters. Tanggal Akses : 1 April 2011.

[BUR-XX] Burton, Bardley F. dan Victor W. Marek. Tanpa Tahun. *Applications of Java Programming Language to Database Management*. Lexington : Department of Computer Science University of Kentucky.

[DUB-06] DuBois, Paul, Stefan Hinz, dan Carsten Pedersen. 2006. *MySQL Certification Study Guide*. USA : Pearson Education.

[FIS-03] Fisher, Maydene, Jon Ellis, Jonathan Bruce. 2003. *JDBC API Tutorial and Reference, Third Edition*. California : Addison Wesley.

[HAI-02] Hainaut, Jean – Luc. 2002. *Introduction to Database Reverse Engineering*. Belgium : Laboratory of Database Application Engineering Institut d'Informatique-University of Namur.

[OKE-09] Okereke, Gerald C., Eco Communication Inc., Lagos Ikeja. 2009. *Database Managemen System*. National Open Univercity of Nigeria.

[LOR-07] Lorentz, Diana, Cathy Shea, dan Simon Watt .*Oracle Database SQL Language Quick Reference*. USA : Oracle USA, Inc.

[MAT-00] Mata – Toledo, Ramon A., dan Pauline K. Chusman. 2000. *Fundamentals of Relational Databases*. USA : Schaum's Outline.

[MAT-05] Matthew, Neil dan Richard Stones. 2005. *Beginning Database with PostgreSQL : From Novice toProfessional, Second Edition*. New York : Apress.

[MYS-05] MySQLAB. 2005. *MySQL Language Reference*. USA : MySQL Press.

[NAT-04] National Information Standards Organization. 2004. *Understanding Metadata*. Bethesda : NISO Press.

[PEL-05] Pelzer, Trudy. 2005. *MySQL 05 Data Dictionary* : *MySQL 5.0 New Features Series – Part 4*. MySQL AB.

[PAR-06] Parsian, Mahmoud. 2006. *JDBC Metadata, MySQL, and Oracle Recipes : A Problem-Solution Approach*. New York : Apress.

[SMI-03] Smith ,Ian. 2003. *Guide to Using SQL: Sequence Number Generator*. USA : Oracle Corporation.

[THE-05] The PostgreSQL Development Group. 2005. *PostgreSQL 8.1.11 Documentation*.California : The PostgreSQL Development Group.

[ZEI-09] Zeis, Chris, Chris Ruel, dan Michael Wessler. 2009. *Oracle 11g For Dummies*. Canada : Wiley Publishing, Inc.