Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK)
Vol.2, No. 2, Oktober 2015, hlm. 143-151

INTEGRASI ALGORITMA K-MEANS DENGAN BAHASA SQL UNTUK KLASTERISASI IPK MAHASISWA (STUDI KASUS: FAKULTAS ILMU KOMPUTER UNIVERSITAS BRAWIJAYA)

Issa Arwani [1]

[1]Fakultas Ilmu Komputer, Universitas Brawijaya
Email:issa.arwani@ub.ac.id

(Naskah masuk: 11 Juni 2015, diterima untuk diterbitkan: 22 Juli 2015)

Abstrak

Secara umum, aplikasi klasterisasi diimplementasikan di luar *DBMS* dengan mengambil data terlebih dahulu dari basisdata untuk disimpan sementara dalam variabel program (misal dalam sebuah *array*), kemudian baru dilakukan proses klasterisasi. Permasalahan waktu dan keamanan dalam pengambilan data dari *DBMS* dan besarnya data yang akan diklasterisasi mendorong metode lain dimana proses klasterisasi bisa langsung dilakukan di *DBMS*. Klasterisasi dilakukan dengan mengintegrasikan algoritma klasterisasi pada *DBMS* menggunakan bahasa *SQL*. Pada penelitian ini difokuskan pada perancangan dan pengimplementasian integrasi algoritma klasterisasi *K-means* pada *Relational DBMS* dengan menggunakan bahasa *SQL*. Proses klasterisasi dilakukan dengan studi kasus data akademik mahasiswa di Fakultas Ilmu Komputer universitas Brawijaya dengan fitur *IPK*, sks tempuh, sks lulus dan semester. Berdasarkan hasil uji coba dataset akademik dengan variasi jumlah dimensi, jumlah klaster dan metode perhitungan jarak yang berbeda, telah didapatkan hasil pengklasteran data dengan benar. Berdasarkan hasil perhitungan kompleksitas waktu untuk tiap tahap implementasi *K-means* menggunakan *SQL* dan tanpa *SQL*, menunjukkan hasil kompleksitas waktu *asimptotik* yang sama dimana tahap menghitung *euclidean distance* membutuhkan kompleksitas waktu yang paling tinggi.

Kata kunci: *Clustering, K-means, SQL, IPK (Indeks Prestasi Kumulatif)*

Abstract

Generally, clustering implemented with taking data from database to be stored temporarily in a program variable (eg, in an array) then continue with clustering process. Direct clustering where the data is stored by integrating the clustering algorithm using the SQL language on the DBMS is proposed. In this study focused on the design and implementation of K-means clustering algorithm on a Relational DBMS using the SQL language. The clustering process carried out with a case study of GPA student in the Faculty of Computer Science University of Brawijaya. Based on results with a variety of dimensions, the number of clusters and different distance calculation methods, has obtained clustering data correctly. Based on time complexity to review each stage of the implementation K - means using SQL and without SQL, showing the same results of asymptotic time complexity where phase euclidean distance still requires the highest time complexity.

Keywords: *Clustering, K-means, SQL, GPA (Grade Point Average)*

1. PENDAHULUAN

Klasterisasi merupakan salah satu cara untuk mengelompokkan data ke dalam kelas - kelas berdasarkan kemiripan data. Secara umum, aplikasi klasterisasi diimplementasikan dengan mengambil data terlebih dahulu dari basisdata untuk disimpan sementara dalam variabel program (misal dalam sebuah *array*), kemudian baru dilakukan proses klasterisasi. Meskipun mengklaster data bisa dilakukan di luar *DBMS (Database Management System)*, tetapi waktu yang dibutuhkan untuk mengambil data pada aplikasi dari basisdata perlu diperhatikan. Sehingga akan lebih efisien proses klasterisasi dilakukan langsung di *DBMS* (Ordonez, 2016). *SQL (Standart Query Language)* adalah bahasa standar, tersedia pada semua *Relational DBMS*. *SQL* akan melindungi aplikasi dari mekanisme internal dalam *DBMS* (Scalzo, 2011). Memanfaatkan dan mengolah data lebih lanjut akan lebih aman dan lebih mudah dilakukan di dalam *DBMS* dengan menggunakan *SQL query* data dari pada diolah di luar *DBMS*.

Indeks Prestasi Kumulatif (IPK) adalah suatu angka yang menunjukkan prestasi mahasiswa dari seluruh mata kuliah yang ditempuh yang dihitung dengan jumlah dari perkalian sks tiap mata kuliah dikalikan dengan bobot nilai yang diperoleh dibagi dengan jumlah sks yang ditempuh. Nilai IPK mahasiswa sering digunakan oleh prodi untuk melakukan berbagai evaluasi akademik. Penting bagi akademik prodi untuk mengetahui persebaran dan pengelompokan IPK dengan metode klasterisasi

untuk melakukan monitoring evaluasi keberhasilan akademik mahasiswa (PTIIK, 2012). Metode klasterisasi yang dilakukan oleh akademik selama ini adalah dengan cara mengunduh terlebih dahulu data dari server baru dilakukan proses klasterisasi dengan aplikasi diluar *DBMS*.

Pada Penelitian ini difokuskan pada perancangan dan pengimplementasian *SQL* untuk merealisasikan integrasi algoritma klasterisasi *K-means* langsung pada *Relational DBMS*. Klasterisasi kemudian diujicobakan pada basisdata akademik mahasiswa FILKOM Universitas Brawijaya untuk pengelompokkan evaluasi keberhasilan akademik mahasiswa dengan fitur IPK, sks tempuh, sks lulus dan semester. Ujicoba dan evaluasi dilakukan dengan membandingkan kesesuaian hasil klasterisasi dan perhitungan kompleksitas waktu yang dilakukan langsung pada *DBMS* menggunakan *SQL* dan di luar *DBMS* menggunakan aplikasi klasterisasi dengan variasi jumlah klaster, jumlah dimensi dan metode rumus perhitungan jarak yang berbeda.

2. DASAR TEORI

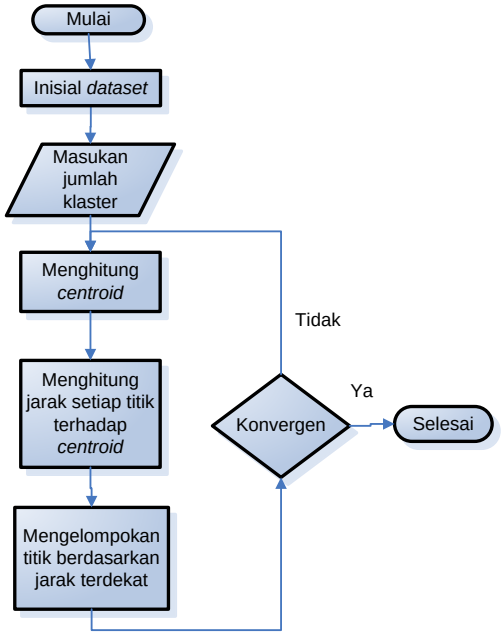
2.1 Klasterisasi

Klasterisasi adalah sebuah proses untuk mengelompokkan data kedalam kelas – kelas berdasarkan kemiripan data (Hung, 2005). Data – data yang memiliki kesamaan pola akan dikelompokkan kedalam satu klaster yang berbeda dengan klaster yang lain. Untuk mengukur kemiripan data digunakan fungsi pengukuran jarak. Semakin dekat jarak antar data semakin mirip pola data tersebut.

Klastrerisasi merupakan sebuah metode metode *unsupervised learning* (Han, 2010). Hal ini disebabkan karena klasterisasi melakukan pengelompokan data tanpa berdasarkan kelas data tertentu. Bahkan klasterisasi dapat dipakai untuk memberikan label pada kelas data yang belum diketahui. Prinsip dari klasterisasi adalah memaksimalkan kesamaan antar anggota satu kelas dan meminimumkan kesamaan antar kelas/klaster. Klasterisasi dapat dilakukan pada data yang memiliki beberapa atribut yang dipetakan sebagai ruang multidimensi.

2.2 Algoritma K-Means

Algoritma *K-means* adalah salah satu algoritma sederhana dari metode *unsupervised learning* yang umum digunakan untuk menyelesaikan permasalahan klasterisasi karena mudah untuk diimplementasikan (Hung dkk.,2005).



Gambar 1 Flowchart Algoritma K-means

Berdasarkan Gambar 1 langkah – langkah algoritma *K-means* (Hung dkk.,2005) akan dijelaskan sebagai berikut :

1) Inisial sebuah *dataset* dari n titik data $X = \{x_1, \ldots, x_n\}$.
2) Masukkan jumlah *k* klaster.
3) Pilih *centroid* C_j untuk *k* klaster dari *dataset*.
4) Tempatkan setiap titik ke klaster terdekat menggunakan pengukuran jarak.
5) Hitung ulang *centroid* dari setiap *k* klaster dengan jumlah data *m* untuk menemukan *centroid* C_j klaster yang baru (V_j), dan hitung jumlah *square error E* yang ditunjukkan dalam rumus (1) dan (2).

$$V_j = \frac{\sum_{i=1}^{n} m(C_j \mid x_i) x_i}{\sum_{i=1}^{n} m(C_j \mid x_i)} \quad \text{untuk } j = 1,\ldots,k \qquad (1)$$

$$E = \sum_{j=1}^{k} \sum_{x_i \prec c_j} \| x_i - v_j \|^2 \quad \text{untuk } i = 1,\ldots n, \; j = 1,\ldots,k \qquad (2)$$

6) Ulangi langkah 3 dan 4 sampai konvergen. Kriteria konvergen : tidak ada lagi penempatan titik-titik data ke klaster baru, perubahan dalam fungsi *error E* berada di bawah *threshold*, atau jumlah iterasi sudah tercapai.

3. PERANCANGAN ALGORITMA *K-MEANS* DENGAN BAHASA *SQL*

Dalam bab ini akan dibahas mengenai perancangan pemetaan algoritma *K-means* kedalam bahasa *SQL*. Secara umum ada dua komponen utama dalam memetakan algoritma *K-means* kedalam bahasa *SQL* yaitu bagaimana mengekspresikan

perhitungan matrik kedalam *SQL* dan mendesain *query* dengan baik untuk mengoptimalkan waktu eksekusinya (Nandagopalan, 2010).

Ada beberapa istilah *dataset* yang direpresentasikan dalam tabel matrik. Input untuk mengimplementasikan algoritma *K-means* adalah *dataset Y* yang terdiri dari *n* titik dengan *d* dimensi. Matrik *C* dan *R* adalah matrik dengan ordo dxk dan matrik *W* adalah matrik dengan ordo *kx1*. Untuk mempermudah pengoperasian dalam mengakses nilai dari masing-masing matrik dibuat inisialisasi *subscript i , j dan k*. Matrik dan *subscript* untuk indek ditunjukkan pada Tabel 1 dan Tabel 2.

Tabel 1 Matrik

Matrik	Ukuran	Isi
Y	*d x n*	*Dataset*
W	*k x 1*	Bobot dari tiap klaster
C	*d x k*	Rata – rata nilai tiap klaster
R	*d x k*	Rata – rata dari kuadrat jarak (*varian*)

Tabel 2 indek

Indek	Rentang Nilai	Digunakan pada
i	*1 .... n*	Titik data
j	*1 .... k*	Klaster
ℓ	*1 .... d*	Dimensi

Langkah – langkah dari algoritma *K-means* dengan bahasa *SQL* secara umum adalah sebagai berikut :

1) Setup, pembuatan, pengindeksan dan pengisian tabel.
2) Inisialisasi, menentukan *centroid* awal pada matrik *C* .
3) Menghitung jarak *k* per titik Y_i. dan mementukan *centroid* terdekat dari masing – masing nilai Y_i.
4) Mengupdate nilai matrik *W,C* dan *R* serta mengupdate tabel model hasil dari pengklasteran *K – means*.
5) Langkah 3-4 diulang hingga pengklasteran *K – means* stabil.

3.1 Setup: pembuatan, pengindeksan dan pengisian tabel

Pada tahap ini akan dilakukan pendefinisian tabel menggunakan statemen *Data Definition Language* (*DDL*). Ada delapan tabel yang akan didefinisikan diawal yaitu tabel *YH(i,*$Y_1,Y_2,\ldots,Y_d$ *), YV(i,* ℓ*,val), CH(j,*$Y_1,Y_2,\ldots,Y_d$ *), C(j,*ℓ*,val), YD(i,j,distance), YNN(i,j), W(j,w), R (j,*ℓ*,val)* dan *model (avg_q, iterasi)*. Kolom yang digarisbawahi untuk masing – masing tabel adalah primary key dari tabel tersebut. Subscript *i,j,*ℓ dan dimensi *d* didefinisikan sebagai integer, jarak,dan nilai matrik *W,C,R* didefinisikan sebagai *float*. Tabel *model* didefinisikan untuk menyimpan bobot *varian* dan jumlah iterasi yang bisa digunakan untuk menentukan kriteria konvergen terpenuhi. Sebelum pengisian data harus dipastikan tabel dalam keadaan kosong. Dibawah ini akan ditunjukkan query yang memberikan nilai *i* secara unik berdasarkan jumlah data selama mengambil data dari tabel *Y* dimana tabel *Y* merupakan tabel yang berisi data yang diambil dari tabel *IPK_MHS* yang berisi data *NIM, nama mahasiswa, semester dan IPK, sks tempuh, sks lulus*.

```
INSERT INTO Y SELECT NIM, MHS, SEM, IPK
SKS_TEMPUH, SKS_LULUS FROM IPK_MHS;
```

Tabel *YH* didefinisikan sebagai tabel "horisontal" dengan *d+1* kolom: *YH(i,*$Y_1,Y_2,\ldots,Y_d$*)* yang mana *i* sebagai *primary key*. Disini dibutuhkan sebuah *query* yang bisa memberikan nilai *i* yang unik untuk masing–masing baris y_i. Di bawah ini akan ditunjukkan *query* yang memberikan nilai *i* secara unik berdasarkan jumlah data selama mengambil data dari tabel *Y*. Kolom yang diambil dari tabel *Y* disesuaikan dengan dimensi data yang ingin diklaster. Berikut contoh query untuk mengisi tabel *YH* dengan 2 dimensi kolom yaitu *SEM* dan *IPK* dari Tabel *Y*.

```
INSERT INTO YH
SELECT sum(1) OVER (rows unbounded
preceding) AS i, SEM,IPK FROM Y
```

Fungsi "*sum()*" dengan klausa "*OVER*" menghitung jumlah yang nilainya dinaikkan sebesar 1 untuk masing – masing titik. Nilai *i* dapat dibuat dengan beberapa fungsi lain di *SQL* yang bisa memberikan nilai unik untuk setiap titik. Tabel *YH* yang didefinisikan diatas tidak cukup digunakan untuk menghitung jarak yang menggunakan fungsi agregasi *SQL* "sum()" dalam proses klasterisasi. Sehingga disini dibutuhkan tabel baru yang didefinisikan sebagai tabel "vertikal" *YV*(i,ℓ,val) yang berisi transformasi nilai *YH* untuk tiap- tiap dimensi dalam tiap-tiap titik. Tabel *YV* diisi dengan pernyataan query sebagai berikut:

```
INSERT INTO YV SELECT i,1,Y1 FROM YH;
....
INSERT INTO YV SELECT i,d,Yd FROM YH;
```

3.2 Inisialisai *Centroid*

Centroid diinisialisasi dengan ditentukan diawal dari tabel *YH*. Disini dibutuhkan tabel *CH*(*j,*$Y_1,Y_2,\ldots,Y_d$) yang digunakan untuk menyimpan nilai *centrtoid* yang ditentukan dari tabel *YH*. Tabel *CH* diisi dengan pernyataan *query* sebagai berikut:

```
INSERT INTO CH SELECT 1,Y1,Y2, ... Yd FROM
YH WHERE i=1;
. . .
INSERT INTO CH SELECT k,Y1,Y2, ... Yd FROM
YH WHERE i=k;
```

Setelah tabel *CH* diisi, tabel *CH* dapat digunakan untuk inisialisasi tabel *C* yang didefinisikan sebagai *C(j,*ℓ*val)*. Tabel *C* ini

digunakan untuk mengakses nilai *centroid* untuk tiap-tiap dimensi dari tabel *CH*. Tabel *C* diisi dengan pernyataan *query* sebagai berikut:

```
INSERT INTO C SELECT k,1,Y1 FROM CH;
....
INSERT INTO C SELECT k,d,Yd FROM CH;
```

3.3 Menghitung jarak dengan *Euclidean Distance*

Pada tahap ini akan dilakukan perhitungan jarak dari tiap-tiap titik terhadap masing – masing *centroid*. Dalam Perhitungan jarak dibutuhkan nilai dari tabel *YV* dan tabel *C* sebgai masukan. Disini dibutuhkan tabel *YD* untuk menyimpan keluaran dari hasil perhitungan jarak untuk masing –masing titik terhadap setiap *centroid*. Tabel *YD* didefinisikan sebagai *YD*(i,j,distance). Tabel *YD* diisi dengan pernyataan *query* sebagai berikut:

```
INSERT INTO YD SELECT i,j,sum(pow
((YV.val -C.val),2))as distance
FROM YV,C WHERE YV.ℓ =C.ℓ GROUP BY i,j;
```

3.4 Menemukan Jarak *Centroid* Terdekat

Pada tahap ini akan ditentukan jarak terdekat untuk setiap titik terhadap terhadap masing-masing *centroid* yang berasal dari tabel *YD*. Hasil perhitungan ini akan disimpan didalam tabel *YNN* yang didefinisikan sebagai *YNN*(i,j). Dalam hal ini dibutuhkan dua tahap dalam *SQL*. Tahap pertama adalah menentukan jarak terdekat. Tahap kedua adalah memberi tanda hasil pengklasteran setiap titik berdasarkan jarak terdekat. Tabel *YNN* diisi dengan pernyataan *query* sebagai berikut:

```
INSERT INTO YNN
SELECT YD.i,YD.j
FROM YD,(SELECT i,min(distance) AS
mindist FROM YD GROUP BY i) YMIND
WHERE YD.i = YMIND.i and YD.distance =
YMIND.mindist;
```

3.5 Mengupdate Hasil Klasterisasi

Langkah selanjutnya yaitu mengupdate tabel *W,C,R* berdasarkan tabel *YNN* pada langkah 3.4. Tabel *W* menyimpan perbandingan jumlah titik terhadap jumlah data untuk masing-masing klaster., tabel *C* mnyimpan nilai *centroid* baru berdasarkan rata-rata nilai titik yang berada pada tiap klaster, dan tabel *R* menyimpan nilai varian berdasarkan nilai centroid yang baru pada tabel *C*. *Query* untuk menyimpan isi dari masing – masing tabel adalah sebagai berikut :

```
INSERT INTO W SELECT j,count(*)
FROM YNN GROUP BY j;

INSERT INTO C
SELECT j, ℓ, avg(YV.val) FROM YV,YNN
WHERE YV.i=YNN.i GROUP BY j, ℓ;

INSERT INTO R
SELECT C.j,C. ℓ ,avg ( pow ((YV.val-
C.val),2))As val FROM C,YV,YNN
WHERE YV.i=YNN.i and YV.ℓ =C.ℓ and
YNN.j=C.j
GROUP BY C.j, C. ℓ;
```

Terakhir yaitu tabel *model* di-*update* untuk menyimpan progres dari proses klasterissi *K-means*. Perubahan tabel *model* ini dilakukan setiap *iterasi* pengklasteran. *Query* untuk merubah isi dari tabel *model* adalah sebagai berikut :

```
UPDATE model
FROM(SELECT sum(W.w*R.val) AS avg_q
FROM R,W WHERE R.j=W.j) avgR
SET avg_q= avgR.avg_q,
iteration= iteration+1;
```

4. IMPLEMENTASI ALGORITMA *K-MEANS* MENGGUNAKAN *SQL* PADA *PL/SQL di DBMS*

Pada bab ini akan diimplementasikan hasil perancangan algoritma *K-means* menggunakan *SQL* dari bab 3 pada *PL/SQL* di *DBMS*. *DBMS* yang digunakan adalah *MySQL*. Setiap tahap perancangan integrasi algoritma *K-means* menggunakan *SQL* akan dimasukkan dalam *Store Procedure*. Kemudian dibuat *Store Procedure* utama untuk memanggil setiap tahap klasterisasi hingga konvergen.

4.1 Setup: pembuatan, pengindeksan dan pengisian tabel

Diasumsikan pada *DBMS* sudah ada semua tabel yang dibutuhkan yaitu tabel *YH(i,Y1,Y2,…,Yd), YV(i, ℓ ,val), CH(j,Y1,Y2,…,Yd), C(j,ℓ ,val), YD(i,j,distance), YNN(i,j), W(j,w),* dan *R (j,ℓ,val)*. Berikut *Store Procedure* untuk mengisi tabel *YH* dan *YV* dengan 2 dimensi kolom yaitu *SEM* dan *IPK* dari Tabel *Y*. Sebelum diisi, tabel *YH* dan *YV* harus dikosongkan terlebih dahulu.

```
CREATE PROCEDURE `sp_YH`()
BEGIN
 TRUNCATE YH; // mengosongkan tabel YH
 INSERT INTO YH
 SELECT id AS i, SEM,IPK FROM Y;
END

CREATE PROCEDURE `sp_YV`()
NO SQL
begin
 TRUNCATE YV; // mengosongkan tabel YV
 INSERT INTO YV SELECT i,1,Y1 FROM YH;
 INSERT INTO YV SELECT i,2,Y2 FROM YH;
End
```

4.2 Inisialisai *Centroid*

Inisialisasi centroid dilakukan dengan cara memilih data dari tabel *YH* untuk dimasukkan kedalam tabel *CH* sesuai dengan jumlah klaster yang diinginkan kemudian dimasukkan dalam tabel *C*. Berikut *Store Procedure* untuk mengisi tabel *CH* dan tabel *C*.

```
CREATE PROCEDURE `sp_CH`(IN `k` INT, IN
`br` INT)
 INSERT INTO CH SELECT k,Y1,Y2 FROM YH
 WHERE i=br

CREATE PROCEDURE `sp_C`()
NO SQL
begin
 TRUNCATE C; // mengosongkan tabel C
 INSERT INTO C SELECT j,1,Y1 FROM CH;
 INSERT INTO C SELECT j,2,Y2 FROM CH;
End
```

4.3 Menghitung jarak dengan *Euclidean Distance*

Dalam Perhitungan jarak dibutuhkan nilai dari tabel *YV* dan tabel *C* sebgai masukan. Disini dibutuhkan tabel *YD* untuk menyimpan keluaran dari hasil perhitungan jarak untuk masing –masing titik terhadap setiap *centroid*. Berikut *Store Procedure* untuk mengisi tabel *YD*.

```
CREATE PROCEDURE `sp_YD`()
Begin
 TRUNCATE YD; // mengosongkan tabel YD
 INSERT INTO YD SELECT i,j,sum(pow
 ((YV.val -C.val),2))as distance FROM
 YV,C WHERE YV. l =C. l GROUP BY i,j;
End
```

4.4 Menemukan Jarak *Centroid* Terdekat

Pada tahap ini akan ditentukan jarak terdekat untuk setiap titik terhadap terhadap masing-masing *centroid* yang berasal dari tabel *YD*. Berikut *Store Procedure* untuk mengisi tabel *YNN*.

```
CREATE PROCEDURE `sp_YNN`()
begin
 TRUNCATE YNN; //mengosongkan tabel YNN
 INSERT INTO YNN
 SELECT YD.i,YD.j
 FROM YD,(SELECT i,min(distance) AS
 mindist FROM YD GROUP BY i) YMIND
 WHERE YD.i = YMIND.i and YD.distance =
 YMIND.mindist;
End
```

4.5 Mengupdate Hasil Klasterisasi

Berikut *Store Procedure* untuk mengisi tabel *W, C, R dan* meng-*update* tabel *model* .

```
CREATE PROCEDURE `sp_hasilKlaster`()
NO SQL
Begin
 TRUNCATE W; //mengosongkan tabel W
 TRUNCATE C; //mengosongkan tabel C
 TRUNCATE R; //mengosongkan tabel R

 INSERT INTO W SELECT j,count(*)
 FROM YNN GROUP BY j;

 INSERT INTO C
 SELECT j, l, avg(YV.val) FROM YV,YNN
 WHERE YV.i=YNN.i GROUP BY j, l;

 INSERT INTO R
 SELECT C.j,C. l ,avg ( pow ((YV.val
 C.val),2))As val FROM C,YV,YNN
 WHERE YV.i=YNN.i and YV.l =C.l and
 YNN.j=C.j
 GROUP BY C.j, C. l;

 UPDATE model
 SET avg_q= (SELECT sum(W.w*R.val) AS
 avg_q
 FROM R,W WHERE R.j=W.j),
 iteration= iteration+1;
end
```

4.6 *Store Procedure* Utama

Store Procedure (Guy, 2007) utama diimplementasikan untuk menjalankan setiap *store procedure* sesuai dengan tahap-tahap dalam algoritma *K-means* hingga konvergen. Kriteria konvergen dalam metode ini dikendalikan dalam tabel *model*. Ketika nilai kolom *avg_q* pada tabel *model* konstan/ tidak berubah pada iterasi beikutnya atau nilai iterasi sudah tercapai maka kriteria konvergen sudah terpenuhi. Berikut *Store Procedure* utama untuk menjalankan setiap tahap *store procedure*.

```
CREATE PROCEDURE `sp_Iterasi`( IN
`iterasi` INT)
BEGIN
 declare x double;
 declare it int;
 set it=0;
 set x=-1;

 CALL `sp_YH`();
 CALL `sp_YV`();
 TRUNCATE CH;
 CALL `sp_CH`();
 CALL `sp_C`();

 UPDATE model set avg_q=0, iteration=0;

 WHILE (x!=(select avg_q From model)AND
 It<=iterasi) // kondisi konvergen
 DO
   CALL `sp_YD`();
   CALL `sp_YNN`();
   SET x=(select avg_q From model);
   CALL `sp_hasilKlaster`();
   set it=it+1;
 END WHILE;
END
```

5. UJI COBA DAN EVALUASI

Uji coba dilakukan dengan tiga buah skenario yang berbeda. Uji coba pertama adalah uji coba verifikasi pengimplemetasian *SQL K-Means* dengan

data dalam skala kecil. Uji coba kedua adalah uji coba validasi pengimplemetasian *SQL K-Means* dengan data akademik FILKOM UB dengan variasi jumlah dimensi, klaster dan rumus jarak. Ujicoba ketiga adalah perhitungan kompleksitas waktu. Evaluasi dilakukan dengan menyimpulkan analisis dari hasil uji coba dan analisis perhitungan kompleksitas waktu.

5.1 Uji Coba Verifikasi

Uji coba pertama adalah uji coba verifikasi pengimplemetasian *K-Means* menggunakan *SQL* dengan data dalam skala kecil. Tabel 3 dan tabel 4 menunjukkan karakteristik dari *dataset* dengan jumlah data 5 baris, 2 dimensi (Y_1 mewakili data semester, Y_2 mewakili data *IPK*) dan 2 klaster. Proses klasterisasi akan dilakukan dalam satu kali iterasi.

Tabel 3 Isi Tabel YH

i	Y_1	Y_2
1	1	2
2	1	2.25
3	1	1.5
4	6	3
5	6	3.25

Tabel 4 Isi Tabel CH

i	Y_1	Y_2
1	1	2.25
2	6	3.25

Hasil uji coba tahap selanjutnya ditunjukkan pada tabel 5 sampai tabel 10.

Tabel 5 Isi Tabel YV

i	ℓ	Val
1	1	1
1	2	2
2	1	1
2	2	2.25
3	1	1
3	2	1.5
4	1	6
4	2	3
5	1	6
5	2	3.25

Tabel 6 Isi Tabel C

j	ℓ	Val
1	1	1
2	1	6
1	2	2.25
2	2	3.25

Tabel 7 Isi Tabel YD

i	j	*Dist*
1	1	0.062
1	2	26.562
2	1	0
2	2	26
3	1	0.562
3	2	28.062
4	1	25.562
4	2	0.0625
5	1	26
5	2	0

Tabel 8 Isi Tabel YNN

i	j
1	1
2	1
3	1
4	2
5	2

Tabel 9 Isi Tabel W

j	w
1	3
2	2

Tabel 10 Isi Tabel Model

Avg_q	*iteration*
0.322	1

5.2 Uji Coba Validasi

Pada metode uji coba validasi, akan dilakukan klasterisasi dengan memasukkan *dataset* dari data akademik mahasiswa FILKOM Universitas Brawijaya sejumlah 4968. Terdapat 2 model *dataset* yang digunakan yaitu data akademik yang berisi data semester dan IPK untuk mewakili *dataset* 2 dimensi, dan data akademik yang berisi semester, IPK, sks beban untuk mewakili *dataset* 3 dimensi. Dari masing-masing *dataset* tersebut akan ditentukan variasi jumlah klaster dan *centroid* awalnya. Dari hasil uji coba, didapatkan persebaran data pada tabel 11 untuk *dataset* 2 dimensi dan tabel 12 untuk *dataset* 3 dimensi dimana dengan variasi jumlah dimensi, jumlah klaster dan metode perhitungan jarak yang berbeda dapat mengklaster data dengan benar.

Tabel 11 Perbandingan hasil klaster untuk *dataset* 2 dimensi

***Dataset* Akademik (IPK)**	**Hasil**	***K-means* pada Aplikasi**	***K-means* menggunakan SQL**
n: 4968	k_1	2361	2361
d:2	k_2	2607	2607
k:2	iterasi	3	3
n: 4968	k_1	1120	1120
d:2	k_2	1487	1487
k:3	k_3	2361	2361
	iterasi	5	5
n: 4968	k	116	116
d:2	k_2	2094	2094
k:4	k_3	1487	1487
	k_4	1271	1271
	iterasi	8	8
n: 4968	k_1	103	103
d:2	k_2	987	987
k:5	k_3	704	704
	k_4	1903	1903
	k_5	1271	1271
	iterasi	7	7

Tabel 12 Perbandingan hasil klaster untuk *dataset* 3 dimensi

***Dataset* Akademik (IPK)**	**Hasil**	***K-means* pada Aplikasi**	***K-means* menggunakan SQL**
n: 4968	k_1	2439	2439
d:3	k_2	2529	2529
k:2	iterasi	5	5
n: 4968	k_1	1416	1416
d:3	k_2	1442	1442
k:3	k_3	2110	2110
	iterasi	4	4
n: 4968	k_1	19	19
d:3	k_2	1300	1300
k:4	k_3	2280	2280
	k_4	1369	1369
	iterasi	3	3
n: 4968	k_1	19	19
d:3	k_2	1042	1042
k:5	k_3	1150	1150
	k_4	1388	1388
	k_5	1369	1369
	iterasi	5	5

Ujicoba validasi juga dilakukan pada proses klasterisasi dengan variasi metode pengukuran jarak (Sasi, 2010). Ada 5 metode pengukuran jarak yang akan dibandingkan dan dievalausi yaitu: *Euclidean distance, Manhattan distance, Standardized Euclidean distance, Max Chebyshev distance,* dan

Minkowski distance. Tabel 13 menunjukkan hasil tren yang sama dari klasterisasi *dataset* dengan variasi jumlah klaster untuk masing-masing metode pengukuran jarak.

Tabel 13 Perbandingan hasil klaster dengan variasi metode pengukuran jarak

Data set	Hasil	*Metode Pengukuran Jarak*				
		Euclidean	***Manhattan***	***Standardized Euclidean***	***Max Chebyshev***	***Minkowski***
n: 4968	k_1	2439	2430	2430	2439	2439
d:3	k_2	2529	2538	2538	2529	2529
k:2	Iterasi	5	3	3	4	5
n: 4968	k_1	1416	1410	1410	1416	1416
d:3	k_2	1442	1444	1444	1439	1442
k:3	k_3	2110	2114	2114	2113	2110
	iterasi	4	4	4	4	4
n: 4968	k_1	19	19	19	19	19
d:3	k_2	1300	1307	1307	1300	1300
k:4	k_3	2280	2273	2273	2280	2280
	k_4	1369	1369	1369	1369	1369
	iterasi	3	4	5	3	3

5.3 Perhitungan Kompleksitas Waktu

Kompleksitas waktu akan diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan *n (T(n))*. Dari perhitungan *T(n)* kemudian ditentukan *"O" (Big-O)* yang merupakan perhitungan kompleksitas waktu *asimptotik* (Sanjeev, 2009).

5.3.1 Kompleksitas waktu Algoritma *K-means* tanpa *SQL*

Berikut akan dijelaskan langkah-perlangkah perhitungan kompleksitas waktu pada algoritma *K-means* tanpa menggunakan *SQL* yang dijalankan pada aplikasi diluar *DBMS*.

1. *Pseudocode* inisial dataset

```
for i ← 1 to n do
  for ℓ ← 1 to d do
       set data[i][ ℓ];
  end for
end for
```

Operasi yang mendasar pada tahap inisial *dataset* adalah inisialisasi *dataset* sebayak *n* dengan *d* dimensi yang diambil dari server database dengan kompleksitas waktu *T(n)=nd*. Kompleksitas waktu *asimptotik* dari $T(n)=nd=O(n^2)$.

2. *Pseudocode* inisial *centroid* awal

```
for j ← 1 to k do
  for ℓ ← 1 to d do
    set centroid [j][ℓ];
  end for
end for
```

Operasi yang mendasar pada tahap inisial *centroid* awal adalah inisialisasi *centroid* sebayak *k* data yang diambil dari *dataset* dengan kompleksitas waktu *T(n)=kd*. Kompleksitas waktu *asimptotik* dari $T(n)=kd=O(k^2)$.

3. *Pseudocode* menghitung *Euclidean Distance*

```
for i ← 1 to n do
 for j ← 1 to k do
  dist ← 0 ;
  for ℓ ← 1 to d do
     dist ← dist + pow(
     data[i][ℓ]-centroid [j][ℓ],2);
  end for
   set YD[i][j] ← sqrt(dist);
 end for
end for
```

Operasi yang mendasar pada tahap menghitung *Euclidean Distance* adalah menghitung jarak dari *n* data untuk masing-masing *k* kluster dan *d* dimensi dengan kompleksitas waktu *T(n)=nkd*. Kompleksitas waktu *asimptotik* dari $T(n)=nkd=O(n^3)$.

4. *Pseudocode* menentukan jarak *centroid* terdekat

```
for i ← 1 to n do
 min ← 1;
 for j ← 2 to k do
   if YD[i][min]>YD[i][j] then
     min←j;
   end if
 end for
 YNN[i]=min;
end for
```

Operasi yang mendasar pada tahap menentukan jarak *centroid* terdekat adalah membandingkan jarak sebayak *n* data untuk masing-masing *k-1* kluster *centroid* dengan kompleksitas waktu *T(n)=n(k-1)*. Kompleksitas waktu *asimptotik* dari $T(n)=nk-n=O(n^2)$.

5. *Pseudocode* meng-*update centroid* baru.

```
for i ← 1 to n do
  j ← YNN[i];
  for ℓ ← 1 to d do
     centroid[j[ℓ]← centroid[j[ℓ]
     + data[i][ℓ];
  end for
  weight[j]++;
end for

for j ← 1 to k do
  for ℓ ← 1 to d do
     centroid[j[ℓ] ← centroid[j[ℓ]
     / weight[j];
  end for
end for
```

Operasi yang mendasar pada tahap meng-*update centroid* baru adalah menghitung jumlah nilai tiap *d* dimensi dari *n* data untuk masing-masing *k* kluster kemudian dibagi bobotnya (*wight[k]*) dengan kompleksitas waktu $T(n)=nd + kd$. Kompleksitas waktu *asimptotik* dari $T(n)=nd+kd=O(n^2)$.

5.3.1 Kompleksitas waktu Algoritma *K-means* dengan *SQL*

Berikut akan dijelaskan langkah-perlangkah perhitungan kompleksitas waktu pada algoritma *K-means* dengan menggunakan *SQL* yang dijalankan pada *DBMS*. Pehitungan kompleksitas waktu dihitung berdasarkan pengaksesan data dari tabel pada setiap tahap rancangan *SQL* pada bab 3.

1. Tahap inisial *dataset*
   Pada tahap inisialisasi *dataset* dilakukan pengisian data ke tabel $YH(\underline{i},Y_1,Y_2,\ldots,Y_d)$ sebanyak *n* baris data dan tabel $YV(\underline{i}, \underline{\ell},val)$ sebanyak *nd* baris data dengan kompleksitas waktu $T(n)=n+nd$. Kompleksitas waktu *asimptotik* dari $T(n)=n+nd=O(n^2)$.
2. Tahap initial *centroid* awal
   Centroid diambil dari tabel $YH (\underline{i},Y_1,Y_2,\ldots,Y_d)$ sebanyak *k* baris data, dan disimpan dalam tabel $CH(\underline{j},Y_1,Y_2,\ldots,Y_d)$. Tabel $C(\underline{\ell},\underline{j},val)$ sebanyak *kd* baris data didefinisikan untuk mengakses nilai centroid untuk tiap-tiap dimensi dari tabel *CH*. kompleksitas waktu waktu $T(n)=k+dk$. Kompleksitas waktu *asimptotik* dari $T(n)=k+kd=O(k^2)$.
3. Tahap menghitung *Euclidean Distance*
   Pada tahap menghitung jarak dengan *euclidean distance* akan dilakukan pada Tabel $YD(\underline{i},\underline{j},distance)$ yang berisi *nk* baris data. Sebelum pernyataan *GROUP BY* pada *SQL* pengisian tabel *YD* dijalankan, dilakukan pengambilan data dari tabel $YV(\underline{i}, \underline{\ell},val)$ yang berisi *nd* baris data yang direlasikan dengan tabel *C* yang berisi *kd* baris data. Kompleksitas waktu pada tahap ini adalah $T(n)=nkd$. Kompleksitas waktu *asimptotik* dari $T(n)=nkd=O(n^3)$.
4. Tahap menentukan jarak *centroid* terdekat
   Pada tahap menentukan jarak *centroid* terdekat dibutuhkan 2 kali pengambilan data dari tabel *YD* yang berisi *nk* baris data untuk menentukan jarak minimum dan hasil pengklusteran untuk setiap poin. Kompleksitas waktu pada tahap ini adalah $T(n)=2nk$. Kompleksitas waktu *asimptotik* dari $T(n)= 2nk =O(n^2)$.
5. Tahap meng-*update centroid* baru
   Pada tahap mengupdate *centroid* baru dibutuhkan mengakses tabel *YNN* (*n* baris data) kemudian direlasikan dengan tabel *YV*(*nd*). Kompleksitas waktu pada tahap ini adalah $T(n)=nd$. Kompleksitas waktu *asimptotik* dari $T(n)= nd =O(n^2)$.

Perbandingan kompleksitas waktu *asimptotik* antara implementasi algoritma *K-means* dengan *SQL* dan algoritma *K-means* pada aplikasi ditunjukkan pada tabel 14 dimana kedua model memberikan perhitungan kompleksitas waktu *asimptotik* yang sama.

Tabel 14 Perbandingan Perhitungan Kompleksitas Waktu

Langkah	*K-means* dengan *SQL*	*K-means* tanpa *SQL*
1. Inisial *dataset*	$O(n^2)$	$O(n^2)$
2. Inisial Centroid awal	$O(k^2)$	$O(k^2)$
3. Menghitung *Euclidean Distance*	$O(n^3))$	$O(n^3)$
4. Menentukan jarak centroid terdekat	$O(n^2)$	$O(n^2)$
5. Meng-*update centroid* baru	$O(n^2)$	$O(n^2)$

5.4 Evaluasi

Setelah melakukan uji coba, perlu dilakukan evaluasi terhadap hasil uji coba tersebut. Hasil evaluasi uji coba adalah sebagai berikut:

1. Pada uji coba verifikasi dimana pengklasteran dilakukan pada *dataset* dengan skala kecil menunjukkan bahwa klasterisasi dengan algoritma *K-means* menggunakan bahasa *SQL* dapat memberikan hasil pengklasteran dengan benar dan sesuai dengan uji coba yang dilakukan secara manual.
2. Pada uji coba validasi, telah dilakukan klasterisasi dengan algoritma *K-means* menggunakan bahasa *SQL* pada *dataset* dengan variasi jumlah dimensi, jumlah klaster dan metode perhitungan jarak yang berbeda dan dapat mengklaster data dengan benar. Hal ini juga dibandingkan dengan pengklasteran *dataset* dengan algoritma *K-means* di luar *DBMS*.
3. Berdasarkan hasil perhitungan kompleksitas waktu untuk tiap tahap implementasi *K-means* menggunakan *SQL* dan tanpa *SQL* menunjukkan hasil kompleksitas waktu *asimptotik* yang sama dimana tahap menghitung *euclidean distance*, membutuhkan kompleksitas waktu yang paling tinggi.

6. KESIMPULAN

Integrasi setiap langkah dalam algoritma *K-means* langsung pada *DBMS* telah dirancang dan diimplementasikan menggunakan bahasa *SQL* yang bisa digunakan untuk klasterisasi data dengan benar.

Berdasarkan hasil uji coba pada *dataset* akademik mahasiswa, proses klasterisasi dengan algoritma *K-means* menggunakan *SQL* menunjukkan hasil data yang sama untuk setiap klasternya jika dibandingkan dengan klasterisasi data dengan aplikasi diluar *DBMS*. Berdasarkan hasil perhitungan kompleksitas waktu untuk tiap tahap implementasi *K-means* menggunakan *SQL* dan tanpa *SQL* menunjukkan hasil kompleksitas waktu *asimptotik* yang sama, dimana tahap menghitung *euclidean distance* membutuhkan kompleksitas waktu yang paling tinggi sehingga perlu dilakukan optimasi *SQL*-nya untuk penelitian selanjutnya.

7. DAFTAR PUSTAKA

Akademik PTIIK. 2012. *Buku Pedoman Pendidikan PTIIK 2012*., UB Press, Malang.

Guy H, 2007, *MySQL Stored Procedure Programming.*, O'Reilly Media, USA.

Han, Jiawei dan Micheline Kamber . 2010. *Data Mining : Concepts and Techniques 3rd edition*., Morgan Kaufmann Publishers, USA.

Hung ,Ming-Chuan, Jungpin Wu, Jin-Hua Chang, dan Don-Lin Yang. 2005. An Efficient k-Means Clustering Algorithm Using Simple Partitioning. *Journal of Information Science and Engineering*, 21, 1157-1177.

Nandagopalan, Adiga, Dhanalakshmi. 2010, A Fast K-Means Algorithm for the Segmentation of Echocardiographic Images Using DBMS-SQL, *IEEE Trans. Knowledge and Data Eng*, 2(2), 162-166.

Ordonez, Carlos. 2004. Programming the *K-means* Clustering Algorithm in SQL. *Proc. ACM Int'l Conf. Knowledge Discovery and Data Mining*, 823-828.

Ordonez, Carlos. 2016. Integrating K-means Clustering with a Relational DBMS Using SQL. *IEEE Trans. Knowledge and Data Eng*., 18(2), 188-201.

Sanjeev. 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press., New York.

Sasi K. Pitchaimalai, Carlos Ordonez, Carlos Garcia-Alvarado, "Efficient distance computation using SQL queries and UDFs," *IEEE International Conference on Data Mining Workshops*, 2008.

Scalzo B. 2011. *Introduction to SQL Server: Basic Skills for Any SQL Server User*. CreateSpace Independent Publishing Platform., USA.