BAB 12
STORED PROCEDURE

12.1. Tujuan Pembelajaran

1. Membuat stored procedure
2. Mengetahui cara kerja stored procedure
3. Mengetahui kegunaan stored procedure

12.2. Dasar Teori

Pengenalan Stored Procedure

Stored Procedure adalah sebuah fungsi berisi kode SQL yang dapat digunakan kembali. DalamStored Procedure juga dapat dimasukkan parameter sehingga fungsi dapat digunakan lebih dinamis berdasarkan parameter tersebut.

Stored procedure merupakan kumpulan dari perintah-perintah SQL yang tersimpan dalam suatu database, pada umumnya digunakan pada perintah DML(*select, insert, update*, dan *delete*). Pada sebuah *stored procedure* dapat diberikan parameter sesuai dengan kebutuhan, sehingga dengan menerapkan *stored procedure* yang disimpan didalam suatu database program aplikasi akan dapat digunakan lebih dinamis berdasarkan parameter yang diberikan, juga *stored procedure* dapat dikombinasikan dengan fungsi-fungsi pernyataan bersyarat dan berkondisi seperti fungsi IF, While.

Apa tujuan dari *stored procedure*? Salah satu tujuan atau kegunaan dari *stored procedure*, yaitu untuk mengefisienkan pembuatan code program pada suatu bahasa pemrograman ketika akan membuat program CRUD(*Create, Read, Update, Delete*) menggunakan database MySQL. *Storedprocedure* dibuat dan disimpan hanya satu kali pada sebuah database, selanjutnya dapat dipanggilkembali, sehingga prosedur dengan nama yang sama dalam program dapat lebih cepat dieksekusi. Jenis perintah-perintah dalam *stored procedure* meliputi menambah data(*insert*),mengubah data (*update*), menghapus data(*delete*), dan memilih/menyeleksi data (*select*).

Keuntungan Stored Procedure

1. Mengurangi lalu lintas jaringan, stored procedure membantu mengurangi lalu lintasjaringan antara aplikasi dan Server MySQL. Karena alih-alih mengirimkan banyak pernyataan SQL yang panjang, aplikasi hanya harus mengirim nama dan parameterprosedur tersimpan.
2. Pusatkan logika bisnis dalam database, kalian dapat menggunakan stored procedure untuk mengimplementasikan logika bisnis yang dapat digunakan kembali oleh beberapa

aplikasi. stored procedure membantu mengurangi upaya duplikasi logika yang sama dibanyak aplikasi dan membuat database Anda lebih konsisten.

3. Jadikan basis data lebih aman, administrator basis data dapat memberikan hak istimewayang sesuai untuk aplikasi yang hanya mengakses stored procedure tertentu tanpa memberikan hak istimewa apa pun pada tabel yang mendasarinya.

Kerugian Stored Procedure

1. Penggunaan sumberdaya, Jika kalian menggunakan banyak prosedur tersimpan, penggunaan memori setiap sambungan akan meningkat secara substansial. Selain itu,penggunaan berlebihan sejumlah besar operasi logis dalam prosedur tersimpan akan meningkatkan penggunaan CPU karena MySQL tidak dirancang dengan baik untuk operasi logis.
2. Penyelesaian masalah, sulit untuk men-debug prosedur tersimpan. Sayangnya, MySQLtidak menyediakan fasilitas apa pun untuk men-debug stored procedure seperti produk database perusahaan lainnya seperti Oracle dan SQL Server.
3. Pemeliharaan, mengembangkan dan memelihara prosedur tersimpan seringkali membutuhkan keahlian khusus yang tidak dimiliki oleh semua pengembang aplikasi. Inidapat menyebabkan masalah dalam pengembangan dan pemeliharaan aplikasi.

Sintak Dasar Stored Procedure

```
DELIMITER $$
CREATE                    PROCEDURE
nama_procedure()BEGIN
   code  sql
END$$
DELIMIT
ER ;
```

Pemanggilan Stored Procedure

```
CALL nama_procedure();
```

12.3. Software

- XAMPP
- MySQL

12.4. Pembelajaran stored procedure

12.4.1 Pembelajaran membuat stored procedure Membuat Membuat Stored Procedure

Pertama, buat dulu tabel yang akan digunakan untuk membuat stored procedure. Sebagai contoh saya membuat procedure GetCustomers untuk menampilkan customerName, city dan phone dari tabel customers.

MySQL memberikan hasil kosong (atau no

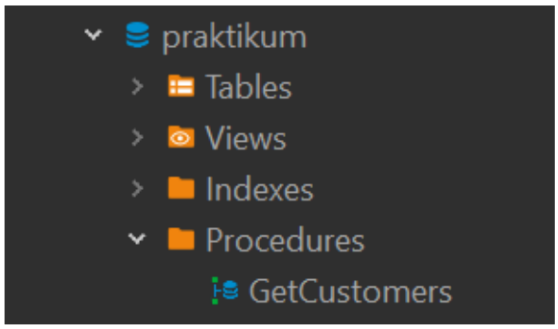
```
1  DELIMITER $$
2
3  CREATE PROCEDURE GetCustomers()
4  BEGIN
5      SELECT
6          customerName,
7          city,
8          phone
9      FROM
10         customers
11     ORDER BY customerName;
12 END$$
13 DELIMITER ;
```

Gambar 12 1 Sintak Dasar Stored Procedure

Kalau sudah membuat stored procedure, silahkan di cek dulu apakah stored procedure sudah ada atau belum. Bila menggunakan xampp bisa dengan melihat menu database di samping kiri > database > procedure.

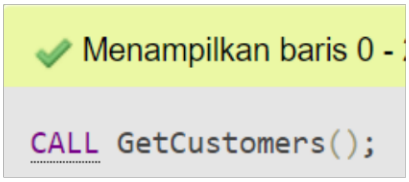
Gambar 12.2 Procedure Yang Tersedia

Bila menggunakan DBeaver, klik database > database kalian > procedures.



Gambar 12.3 Mengecek Menggunakan DBeaver

Setelah selesai menyimpan stored procedure, kalian dapat memanggilnya dengan menggunakan pernyataan CALL:



Gambar 12.4 Memanggil Procedure GetCustomers

Maka outputnya akan menampilkan:

customerName	city	phone
dani	Bogor	546451235697
nino	Cipanas	023145678520
raihan	Bogor	021212457865

Gambar 12.5 Tampilan Procedure GetCustomers

Dan pernyataan tersebut mengembalikan hasil yang sama dengan kueri. Saat pertama kali kalian menjalankan stored procedure, MySQL mencari nama dalam katalog database, mengkompilasi kode stored procedure, menempatkannya di area memori yang dikenal sebagai cache, dan menjalankan stored procedure.

Jika kalian menjalankan stored procedure yang sama di sesi yang sama lagi, MySQL hanya menjalankan prosedur tersimpan dari cache tanpa harus mengkompilasi ulang.

Stored procedure dapat memiliki parameter sehingga kalian dapat memberikan nilai padanya dan mendapatkan hasilnya kembali. Misalnya, kalian dapat memiliki stored procedure yang mengembalikan customers berdasarkan kota dan nomor telepon. Dalam hal ini, kota dannomor telepon adalah parameter dari stored procedure. Stored procedure dapat memanggil stored procedure lain atau fungsi tersimpan, yangmemungkinkan kalian untuk memodulasi kode kalian.

Membuat stored Procedure Dengan Parameter

Seringkali, stored procedure memiliki parameter. Parameter membuat stored procedure lebih berguna dan dapat digunakan kembali. Parameter dalam stored procedure memiliki salah satudari tiga mode: IN,OUT, atau INOUT.

IN Parameter

IN adalah mode default. Saat kalian menentukan parameter IN dalam stored procedure, program pemanggil harus meneruskan argumen ke stored procedure. Selain itu, nilai parameter IN dilindungi. Ini berarti bahwa meskipun kalian mengubah nilai parameter IN di dalam stored procedure, nilai aslinya tidak berubah setelah stored procedure berakhir. Dengan kata lain, stored procedure hanya berfungsi pada salinan parameter IN.

OUT Parameter

Nilai parameter OUT dapat diubah di dalam stored procedure dan nilai barunya dikembalikan ke program pemanggil. Perhatikan bahwa stored procedure tidak dapat mengakses nilai awal parameter OUT saat dimulai.

INOUT Parameter

Parameter INOUT adalah kombinasi dari parameter IN dan OUT. Ini berarti bahwa program pemanggil dapat meneruskan argumen, dan stored procedure dapat mengubah parameter INOUT, dan meneruskan nilai baru kembali ke program pemanggil.

Menentukan parameter

Berikut adalah sintaks dasar untuk mendefinisikan parameter dalam stored procedure:

```
[IN | OUT | INOUT] parameter_name datatype[(length)]
```

Gambar 12.6 Sintak Dasar Procedure dengan Parameter

Dalam sintaks ini:

1. Pertama, tentukan mode parameter, yang bisa IN , OUT atau INOUT tergantung padatujuan parameter dalam stored procedure.
2. Kedua, tentukan nama parameternya. Nama parameter harus mengikuti aturan penamaannama kolom di MySQL.
3. Ketiga, tentukan tipe data dan panjang maksimum parameter.

MySQL Stored Procedure Dengan Parameter

Beberapa contoh penggunaan parameter stored procedure.

Contoh parameter IN

Contoh berikut membuat stored procedure yang menemukan semua detail pembelian yang nilainya ditentukan oleh parameter masukan price:

MySQL memberikan hasil kosong (atau nol

```
1 CREATE PROCEDURE GetPriceMore(
2     IN price INT(5)
3 )
4 BEGIN
5     SELECT *
6     FROM orderdetails
7     WHERE priceEach = price;
8 END;
```

Gambar 12.7 Query Membuat Procedure GetPriceMore

Keterangan kode diatas yaitu, kita membuat sebuah prosedur dengan nama GetPriceMore yang dimana kita membuat parameter penampung yang bernama price, untuk menampilkan semua keterangan di tabel order details.

Jika sudah kita panggil prosedur GetPriceMore seperti ini:

Menampilkan baris 0 - 0 (total 1, Per

```
1 CALL GetPriceMore(60000);
```

Gambar 12.8 Memanggil Procedure GetPriceMore dengan Parameter

Maka outputnya akan menampilkan seperti ini:

orderNumber	productCode	quantityOrdered	priceEach
55555	pia43	50	60000

Gambar 12.9 Tampilan Procedure GetPriceMore

Jika kita ingin mengecek isi di tabel customersdetails apakah sesuai dengan data di prosedur GetMorePrice, silahkan kalian lihat gambar dibawah ini:

orderNumber	productCode	quantityOrdered	priceEach
11111	cnm08	50	100000
22222	fkj45	20	40000
33333	oqh21	50	10000
44444	oiw85	25	20000
55555	pia43	50	60000

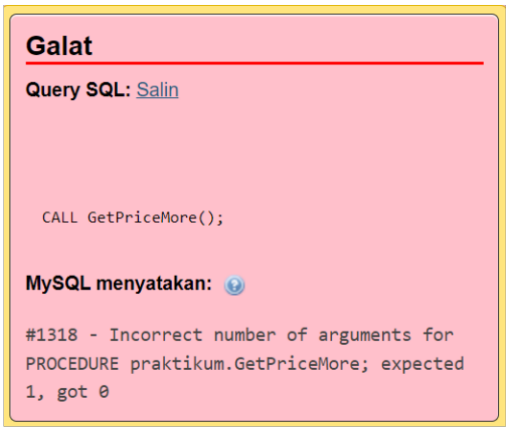
Gambar 12.10 Tampilan Tabel CustomersDetails

Data yang ditampilkan hanya satu record karena kita membuat prosedur dengan nilai ketika **priceEach nilainya sama dengan price,** karena di tabel orderdetails hanya memiliki satu record yang bernilai 60000 maka hanya akan menampilkan satu record di prosedur GetMorePrice.

Karena price adalah parameter IN, maka kalian harus memberikan argumen. Jika kalian tidakmemberi argumen, maka SQL akan menampilkan pesan error:

```
CALL GetPriceMore();
```

Gambar 12.11 Memanggil Procedure GetPriceMore



Gambar 12.12 Error Karena Miss Parameter

Error diatas berarti kita belum menambahkan argument, argumen yang diminta itu sebanyak1 argument.

Contoh parameter OUT

Stored procedure berikut mengembalikan jumlah pesanan berdasarkan status pesanan.

```
DELIMITER $$

CREATE PROCEDURE GetOrderCountByStatus (
    IN  orderStatus VARCHAR(25),
    OUT total INT
)
BEGIN
    SELECT COUNT(orderNumber)
    INTO total
    FROM orders
    WHERE status = orderStatus;
END$$

DELIMITER ;
```

Gambar 12.13 Membuat Procedure Menggunakan Parameter OUT

Stored procedure GetOrderCountByStatus() memiliki dua parameter:

1. OrderStatus: adalah parameter IN yang menentukan status pesanan yang akan dikembalikan.
2. Total: adalah parameter OUT yang menyimpan jumlah pesanan dalam status tertentu.

Untuk mengetahui jumlah pesanan yang telah dikirim, kalian bisa memanggilGetOrderCountByStatus dan meneruskan status pesanan sebagai terkirim, dan jugameneruskan variabel sesi (@total) untuk menerima nilai kembalian.

```
CALL GetOrderCountByStatus('terkirim',@total);
SELECT @total;
```

Gambar 12.14 Memanggil Procedure GetOrderCountByStatus

Maka outputnya akan menampilkan:

@total
4

Gambar 12.15 Tampilan GetOrderCountByStatus

Silahkan lihat tabel orders dibawah ini untuk menyesuaikan data dengan yang ditampilkanoleh total.

orderNumber	customerNumber	orderDate	comment	status
11111	12345	2022-11-02	Bagus	terkirim
22222	23456	2022-11-17	Memuaskan	dikemas
33333	64636	2022-11-05	Jelek	terkirim
44444	95567	2022-11-29	Memuaskan	dikemas
55555	29384	2022-11-25	Bagus	perjalanan
93845	73845	2022-11-22	Memuaskan	terkirim
98765	3948	2022-12-01	excellent	terkirim

Gambar 12.16 Tampilan Tabel Order

Data diatas menampilkan output sebanyak 4 untuk status sebagai terkirim. Dan bila kita lupa atau sengaja tidak mengisi argument, maka akan menampilkan error seperti ini.

```
CALL GetOrderCountByStatus();
```

Gambar 12.17 Memanggil Procedure GetOrderCountByStatus

Galat

Query SQL: Salin

```
CALL GetOrderCountByStatus();
```

MySQL menyatakan:

```
#1318 - Incorrect number of arguments for PROCEDURE praktikum.GetOrderCountByStatus; expected 2, got 0
```

Gambar 12.18 Missing Parameter

Keterangan error di atas yaitu kita belum memasukkan argumen yang diminta, argument disini diminta sebanyak 2 buah, untuk mengetahui berapa argumen yang diminta coba dilihatexpected number nya.

Contoh parameter INOUT

```
1  DELIMITER $$
2
3  CREATE PROCEDURE SetCounter(
4      INOUT counter INT,
5      IN inc INT
6  )
7  BEGIN
8      SET counter = counter + inc;
9  END$$
10
11 DELIMITER ;
```

Gambar 12.19 Query Membuat Procedure SetCounter

Dalam contoh ini, stored procedure SetCounter() menerima satu parameter INOUT (counter) dan satu parameter IN (inc). Ini meningkatkan penghitung (counter) dengan nilai yang ditentukan oleh parameter inc.

Pernyataan ini menggambarkan cara memanggil stored procedure SetCounter:

```
CALL SetCounter(@counter,5);
SELECT @counter;
```

Gambar 12.20 Memanggil Procedure SetCounter dengan Parameter

Maka akan menampilkan output seperti ini:

@counter
NULL

Gambar 12.21 Tampilan Output Procedure SetCounter

Nilai null didapat karena kita belum melakukan inisialisasi atau pemberian nilai terhadap variabel counter, coba kita beri nilai counter nya terlebih dahulu.

```
SET @counter = 1;
CALL SetCounter(@counter,1); -- 2
CALL SetCounter(@counter,1); -- 3
CALL SetCounter(@counter,5); -- 8
SELECT @counter;
```

Gambar 12.22 Memanggil dan Menampilkan Procedure SetCounter

Jika sudah, coba jalankan dan kalian akan mendapat output seperti ini:

@counter
8

Gambar 12.23 Tampilan Output Procedure SetCounter

Penjelasan mengapa mendapat angka 8, pertama kita memberi nilai awal untuk variabel counter sebesar 1, lalu kita panggil store procedure SetCounter(@counter,1).

Berarti kita memberi perintah kepada SQL, untuk melakukan penghitungan (1,1) yang nantinya akan dijumlahkan. Sederhana nya @counter sekarang sudah bernilai 1 makanya didalam kurung nilainya 1.

Selanjutnya bisa dilihat di stored procedure yang kita buat, disitu ada counter = counter + inc.Argumen yang pertama kita dapat yaitu 1 dan ditambah dengan argumen yang kedua yaitu 1,maka nilai counter sekarang menjadi 2. Langkah selanjutnya yaitu kita panggil lagi stored procedure nya, dan kita memberi nilai increment (inc) dengan nilai 1. Berarti SQL membaca nya (2,1) yang berarti counter = 2 + 1 = 3, maka sekarang counter mempunyai

angka 3.

Langkah yang terakhir yaitu kita panggil lagi stored procedure nya, dan kita memberi nilai increment (inc) dengan nilai sebesar 5. Berarti SQL membaca argumen kita seperti ini(3,5) yang berarti counter = 3 + 5 = 8.

Lalu kita memberi perintah select untuk menampilkan counter yang tersimpan mempunyai angka sebesar 8.

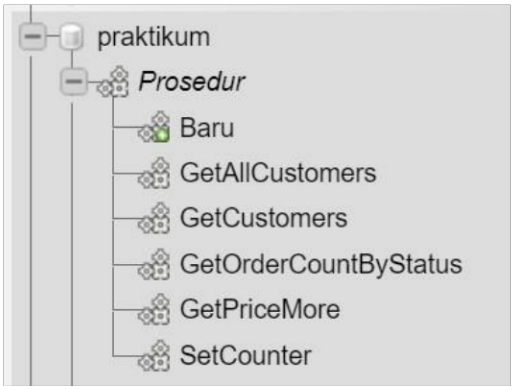
Drop Stored Procedure

Misalnya kita membuat satu store procedure lagi dengan nama GetAllCustomers dimanaakan menampilkan semua informasi di tabel customers.

```
1  DELIMITER $$
2
3  CREATE PROCEDURE GetAllCustomers()
4  BEGIN
5      SELECT
6          customerNumber,
7          customerName,
8          city,
9          phone
10     FROM customers;
11 END$$
12
13 DELIMITER ;
```

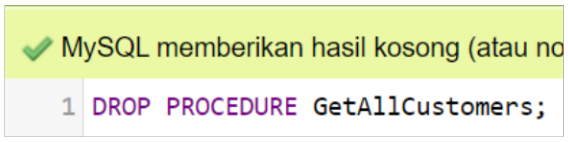
Gambar 12.24 Membuat Procedure GetAllCustomers

Coba di cek apakah sudah ada, kalian bisa mengecek dari menu di sebelah kiri database >procedure > procedure yang sudah dibuat.



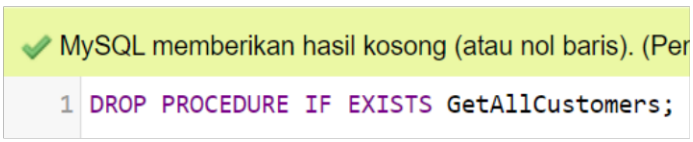
Gambar 12.25 List Procedure

Misalnya saya ingin menghapus procedure GetAllCustomers, kita bisa menghapusnyadengan perintah:



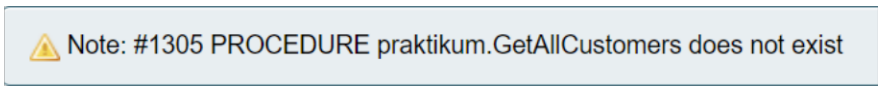
Gambar 12.26 Menghapus Procedure GetAllCustomers

Kalian juga bisa menghapusnya seperti ini:



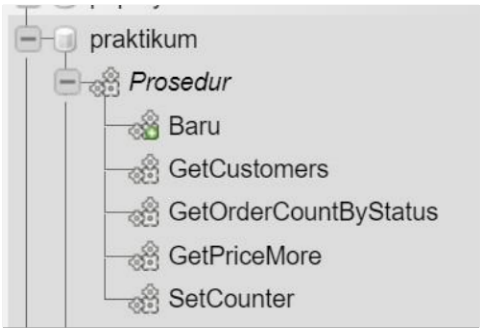
Gambar 12.27 Opsi Lain Menghapus Procedure

Maka outputnya akan menampilkan:



Gambar 12.28 Tampilan Procedure GetAllCustomers Tidak Tersedia

Bila sudah dihapus, coba dicek lagi di menu sebelah kiri database > procedure.



Gambar 12.29 List Procedure Setelah Dihapus Procedure GetAllCustomers

Jika procedure GetAllCustomers sudah tidak ada artinya kita sudah berhasil menghapusprocedure yang kita buat.

12.5. Latihan

1. Buatlah sebuah stored procedure dengan nama GetPriceLarge dengan kondisi dimanapembelian (priceEach) di tabel orderdetails lebih dari 50000!
2. Apakah saat menghapus prosedur diperlukan parameter? Jelaskan alasannya!

Latihan tambahan untuk yang telat:

1. Buatlah sebuah tabel baru bernama kelas, dengan atribut seperti ini:

#	Nama	Jenis
1	**id_kelas** 🔑	int(3)
2	**ruangan**	varchar(10)
3	**hari**	varchar(6)
4	**sesi**	enum('pagi', 'karyawan')

   Isilah datanya dengan minimal 5 record! (Contoh seperti gambar dibawah ini)

id_kelas	ruangan	hari	sesi
123	TI-201	Rabu	pagi
135	TI-301	Senin	karyawan
234	TI-205	Sabtu	karyawan
357	TI-203	Jumat	pagi
456	TI-401	Rabu	pagi

2. Tambahkan kolom baru di tabel mahasiswa bernama kelas, isi sesuai dengan ruangan ditabel kelas! (Contoh seperti gambar dibawah ini)

   Sebelum ditambah kolom ruangan:

NPM	nama_mahasiswa	kelas	alamat	no_telp	semester
202310001	Rana	5-TI-PA	Jalan Padasuka	2147483647	5
202310002	Rani	1-TI-PA	Jalan Paledang	2147483647	1
202310003	Ranu	2-TI-PA	Jalan Jasinga	2147483647	2
202310004	Rane	3-TI-PA	Jalan Rancamaya	2147483647	3
202310006	Rina	7-TI-PA	Jalan Kalimati	2147483647	7
202310007	Rini	8-TI-PA	Jalan Manggis	2147483647	8
202310008	Rinu	4-TI-PA	Jalan Mawar	2147483647	4
202310009	Rine	3-TI-PA	Jalan Kenanga	2147483647	3
202310010	Rino	2-TI-PA	Jalan Ahmad Yani	2147483647	2
202310034	Raihan dwi pratama	5-TI-PA	Jalan Sakura	2147483647	5

Sesudah ditambah kolom ruangan, dan diisi dengan nilai yang sesuai:

NPM	nama_mahasiswa	kelas	alamat	no_telp	semester	ruangan
202310001	Rana	5-TI-PA	Jalan Padasuka	2147483647	5	TI-301
202310002	Rani	1-TI-PA	Jalan Paledang	2147483647	1	TI-201
202310003	Ranu	2-TI-PA	Jalan Jasinga	2147483647	2	TI-203
202310004	Rane	3-TI-PA	Jalan Rancamaya	2147483647	3	TI-205
202310006	Rina	7-TI-PA	Jalan Kalimati	2147483647	3	TI-205
202310007	Rini	8-TI-PA	Jalan Manggis	2147483647	8	TI-401
202310008	Rinu	4-TI-PA	Jalan Mawar	2147483647	8	TI-401
202310009	Rine	3-TI-PA	Jalan Kenanga	2147483647	3	TI-205
202310010	Rino	2-TI-PA	Jalan Ahmad Yani	2147483647	2	TI-203
202310034	Raihan dwi pratama	5-TI-PA	Jalan Sakura	2147483647	5	TI-301

3. Lakukan join dari tabel detail mahasiswa dengan tabel kelas, dengan kondisi kalau ruangan di detail mahasiswa sama dengan ruangan di tabel kelas. Dan tampilkan namamahasiswa, kelas, semester, dan ruangan! (Contoh seperti gambar dibawah ini)

nama_mahasiswa	kelas	semester	ruangan
Rana	5-TI-PA	5	TI-301
Rani	1-TI-PA	1	TI-201
Ranu	2-TI-PA	2	TI-203
Rane	3-TI-PA	3	TI-205
Rina	7-TI-PA	3	TI-205
Rini	8-TI-PA	8	TI-401
Rinu	4-TI-PA	8	TI-401
Rine	3-TI-PA	3	TI-205
Rino	2-TI-PA	2	TI-203
Raihan dwi pratama	5-TI-PA	5	TI-301

Ket: disini semua kondisi terpenuhi, jadi tidak ada nilai null

4. Tampilkan jumlah mahasiswa, semester dan ruangan kelasnya yang dimana sedangmenempuh semester 5!

jumlah_mahasiswa	semester	ruangan
2	5	TI-301

5. Tampilkan jumlah mahasiswa yang dimana sedang menempuh semester 3 dan ruangankelasnya!

mahasiswa	ruangan
3	TI-205