

LAPORAN RESMI
MODUL III
STORED PROCEDURE
SISTEM MANAJEMEN BASIS DATA



NAMA	: ALIYUL RIDHO
N.R.P	: 230441100135
DOSEN	: FITRI DAMAYANTI, S.Kom., M.Kom.
ASISTEN	: ABDUL JABBAR RAMADHANI
TGL PRAKTIKUM	: 19 April 2025

Disetujui : April 2025
Asisten

Abdul Jabbar Ramadhani
21.04.411.00062



LABORATORIUM TEKNOLOGI INFORMASI
PRODI SISTEM INFORMASI
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS TRUNOJOYO MADURA

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam pengelolaan basis data, efisiensi, keamanan, dan standarisasi proses menjadi faktor penting dalam menjaga kualitas sistem informasi. Salah satu fitur yang sangat membantu dalam mendukung hal tersebut adalah *Stored procedure*. *Stored procedure* merupakan sekumpulan perintah SQL yang disimpan dan dijalankan di dalam sistem manajemen basis data (SMBD). Fitur ini memungkinkan pengguna atau developer untuk menyimpan logika bisnis di dalam database, sehingga dapat digunakan kembali tanpa harus menuliskannya secara berulang.

Penggunaan *Stored procedure* memberikan berbagai keuntungan, seperti penyederhanaan proses pemrograman, peningkatan performa karena eksekusi dilakukan di sisi server, serta peningkatan keamanan karena dapat membatasi akses langsung ke data. Selain itu, *Stored procedure* juga mendukung pengelolaan transaksi dan penanganan error yang lebih terstruktur, sehingga sangat bermanfaat dalam pengembangan aplikasi yang membutuhkan interaksi kompleks dengan basis data.

Dalam konteks praktikum, pemahaman dan penerapan *Stored procedure* sangat penting agar mahasiswa mampu mengembangkan sistem yang efisien, terstruktur, dan mudah dipelihara. Dengan menggunakan *Stored procedure*, proses pengolahan data menjadi lebih terstandarisasi dan dapat diintegrasikan secara optimal ke dalam aplikasi yang sedang dikembangkan.

1.2 Tujuan

- Mampu memahami konsep dasar *Stored procedure* didalam basis data
- Mampu memahami penerapan *Stored procedure*

BAB II

DASAR TEORI

2.1 *Stored procedure*

2.1.1 Pengertian *Stored procedure*

Stored procedure adalah sekumpulan perintah SQL yang disimpan dalam database dan dapat dipanggil untuk dieksekusi kapan saja. Fungsinya mirip dengan fungsi (function) dalam bahasa pemrograman, namun dijalankan di sisi server basis data. *Stored procedure* dapat menerima parameter, melakukan proses logika, serta mengembalikan hasil berupa nilai atau manipulasi data.

2.1.2 Jenis – jenis *stored procedure*

Dalam pengelolaan basis data yang kompleks dan dinamis, efisiensi serta konsistensi dalam pengolahan data menjadi faktor penting yang harus diperhatikan. Salah satu solusi yang umum digunakan untuk memenuhi kebutuhan tersebut adalah *stored procedure*. *Stored procedure* merupakan sekumpulan perintah SQL yang disimpan di dalam sistem manajemen basis data dan dapat dijalankan berulang kali secara terstruktur. Dengan memanfaatkan *stored procedure*, proses manipulasi dan pengambilan data dapat dilakukan dengan lebih cepat, aman, dan konsisten, karena logika bisnis yang dibutuhkan telah terenkapsulasi dalam prosedur tersebut. Hal ini tidak hanya mempermudah pengembangan aplikasi, tetapi juga meningkatkan integritas data serta mengurangi duplikasi kode pada sisi klien. Oleh karena itu, penggunaan *stored procedure* menjadi praktik yang sangat direkomendasikan dalam pembangunan sistem informasi berbasis database.

Berikut ini beberapa jenis *stored procedure* :

a) *Stored procedure* Tanpa Parameter

Stored procedure tanpa parameter adalah jenis paling dasar yang berfungsi menjalankan sekumpulan perintah SQL statis, tanpa memerlukan masukan dari luar. Jenis ini sangat berguna ketika perintah yang dijalankan bersifat tetap, seperti menampilkan seluruh isi tabel atau melakukan pembersihan log. Meskipun sederhana, *stored procedure* ini mampu meningkatkan efisiensi karena dapat dipanggil berulang kali tanpa mengetik ulang SQL-nya. Contoh penggunaannya misalnya dalam sistem monitoring yang rutin menampilkan data tertentu tanpa kriteria khusus.

b) *Stored procedure* dengan Parameter IN (*Input*)

Stored procedure dengan parameter IN dirancang untuk menerima data dari pemanggil. Ini menjadikannya jauh lebih fleksibel, karena prosedur dapat menyesuaikan proses berdasarkan nilai *input* yang diberikan. Parameter IN sangat umum digunakan dalam query bersyarat, seperti pencarian berdasarkan ID, tanggal, nama, atau kategori. Dalam praktiknya, ini membantu menghindari hardcoding dan membuat sistem menjadi modular dan mudah dikelola. Misalnya, dalam sistem penjualan, kita bisa membuat prosedur untuk menampilkan semua pesanan berdasarkan *customer_id*, cukup dengan satu prosedur yang bisa digunakan siapa pun di aplikasi.

c) *Stored procedure* dengan Parameter OUT (*Output*)

Berbeda dengan IN, parameter OUT digunakan untuk mengembalikan nilai dari prosedur ke pemanggil. Ini sangat berguna untuk menyimpan hasil dari perhitungan atau pemrosesan di dalam prosedur dan kemudian mengambilnya di luar. Penggunaan OUT membuat prosedur lebih dari sekadar alat eksekusi SQL—ia juga menjadi penghasil nilai. Contohnya, menghitung total pesanan pelanggan dan mengembalikannya ke pemanggil aplikasi agar bisa ditampilkan di UI. Dengan OUT, logika pemrosesan tetap berada di database, sehingga mengurangi beban pada aplikasi dan menjaga keamanan logika bisnis.

d) *Stored procedure* dengan Parameter IN dan OUT

Stored procedure ini menggabungkan kekuatan parameter IN dan OUT, menjadikannya ideal untuk logika bisnis yang kompleks. Kita dapat mengirim *input* ke prosedur, memproses data berdasarkan *input* tersebut, dan mengembalikan hasilnya ke pemanggil. Misalnya, dalam sistem inventori, prosedur bisa menerima ID produk (IN), menghitung sisa stok di gudang, lalu mengembalikan nilainya (OUT). Ini memungkinkan prosedur digunakan sebagai alat bantu utama dalam pengambilan keputusan bisnis, seperti apakah *restock* diperlukan atau tidak.

2.1.3 Syntax *stored procedure*

a) *stored procedure* (tanpa parameter)

```
DELIMITER //
```

```
CREATE PROCEDURE getAllProducts()  
BEGIN  
    SELECT * FROM products;  
END //
```

```
DELIMITER ;
```

Stored procedure jenis ini digunakan ketika kita ingin menjalankan sekumpulan perintah SQL tanpa perlu memberikan *input* apa pun. Misalnya, jika kita ingin menampilkan semua data dari tabel *products*, kita bisa membuat prosedur *getAllProducts()* yang hanya berisi perintah *SELECT * FROM products*;. Saat dipanggil dengan *CALL getAllProducts()*;, prosedur ini akan menjalankan perintah tersebut dan mengembalikan seluruh data tanpa perlu masukan tambahan.

Cara memanggilnya :

```
CALL getAllProducts();
```

b) *stored procedure* (dengan parameter IN)

```
DELIMITER //
```

```
CREATE PROCEDURE getCustomerById(IN cust_id INT)  
BEGIN  
    SELECT * FROM customers WHERE id = cust_id;  
END //
```

```
DELIMITER ;
```

Stored procedure dengan parameter *input* digunakan saat kita ingin mengirimkan nilai dari luar ke dalam prosedur. Contohnya, jika kita ingin mengambil data dari customer berdasarkan ID-nya, kita bisa membuat prosedur *getCustomerById(IN cust_id INT)*. Di dalam prosedur, nilai *cust_id* akan

digunakan dalam klausa WHERE untuk memfilter data yang ditampilkan. Pemanggilannya dilakukan dengan CALL getCustomerById(1);, di mana angka 1 adalah ID customer yang ingin kita cari.

Cara pemanggilannya :

```
CALL getCustomerById(3);
```

c) *Stored procedure* (dengan paramerter)

```
DELIMITER //
CREATE PROCEDURE getProductCount(OUT total INT)
BEGIN
    SELECT COUNT(*) INTO total FROM products;
END //
DELIMITER ;
```

Stored procedure dengan parameter output digunakan ketika kita ingin prosedur mengembalikan nilai ke pemanggilnya, bukan hanya sekedar menampilkan data. Misalnya, prosedur getProductCount(OUT total INT) digunakan untuk menghitung jumlah produk dan menyimpannya dalam variabel total. Saat prosedur ini dipanggil dengan CALL getProductCount(@jumlah);, hasilnya akan disimpan ke dalam variabel @jumlah, yang kemudian bisa ditampilkan dengan SELECT @jumlah

Cara pemanggilannya :

```
CALL getProductCount(@jumlah);
SELECT @jumlah;
```

d) *Stored procedure* (dengan paramerter *IN and OUT*)

```
DELIMITER //

CREATE PROCEDURE checkStock(
    IN product_id INT,
    OUT stock INT
)
//
```

```
BEGIN  
    SELECT quantity INTO stock  
    FROM products  
    WHERE id = product_id;  
END //  
  
DELIMITER ;
```

Prosedur ini menggabungkan parameter *input* dan output, yang memungkinkan kita mengirim data ke dalam prosedur dan sekaligus mendapatkan hasil kembali. Sebagai contoh, prosedur checkStock(IN product_id INT, OUT stock INT) menerima ID produk sebagai *input*, lalu mengembalikan jumlah stok produk tersebut melalui parameter output stock. Prosedur ini dapat dijalankan dengan CALL checkStock(5, @stok);, lalu hasil stoknya bisa dilihat dengan SELECT @stok;.

Cara pemanggilannya :

```
CALL checkStock(3, @stok);  
SELECT @stok;
```


BAB III
TUGAS PERIDATULIAN

12/25
04

3.1 Soal

1. Jelaskan bagaimana cara menangani kesalahan (Error handling) dalam stored procedure. Apa yang harus dilakukan jika terjadi kesalahan saat eksekusi?
2. Sebutkan dan jelaskan keuntungan menggunakan stored procedure dibandingkan dengan query SQL biasa. Apa saja alasan yang mendasari pemilihan stored procedure?
3. Dalam konteks stored procedure, apa saja yang dimaksud dengan "parameter default". Berikan contoh bagaimana cara menetapkan nilai default untuk parameter dalam stored procedure!
4. Jelaskan bagaimana stored procedure dapat berinteraksi dengan transaksi (transaction) dalam basis data. Apa yang perlu diperhatikan saat menggunakan stored procedure dalam transaksi?
5. Apa yang terjadi jika sebuah stored procedure dihapus dari basis data? Apakah data yang dihasilkan oleh stored procedure tersebut akan hilang? Jelaskan alasannya!
6. Berikan contoh situasi dimana menggunakan stored procedure dapat meningkatkan performa aplikasi basis data. Sertakan alasan mengapa stored procedure lebih efisien dalam konteks tersebut!

3.2 Jawaban.

1. Untuk menangani kesalahan dalam stored procedure, digunakan blok BEGIN... EXCEPTION... END. Jika terjadi kesalahan saat eksekusi, prosedur dapat menangkapnya dan memberikan pesan atau rollback transaksi untuk mencegah data rusak.
2. - lebih cepat karena disimpan di server dan dikompilasi sekali, meningkatkan keamanan karena bisa membatasi akses langsung tabel, memudahkan pemeliharaan karena logika terpusat. Alasan memilihnya: efisiensi, keamanan, dan pemeliharaan yang mudah.
3. 'parameter default' adalah nilai awal yang digunakan jika parameter tidak diberikan. contoh: CREATE PROCEDURE GET UIR (@Status VARCHAR(10)) AS BEGIN SELECT * FROM UIR WHERE Status = @Status END.
4. Stored procedure bisa digunakan untuk mengelola transaksi dengan perintah BEGIN TRANSACTION, COMMIT, dan ROLLBACK. penting untuk memastikan semua langkah berhasil agar data konsisten, dan lakukan ROLLBACK jika ada kesalahan.
5. data tidak hilang, data tetap ada karena disimpan di tabel bukan stored procedure. Yang hilang hanya logika yang dibuat oleh procedure tersebut.
6. Aplikasi E-commerce yang sering memproses pesanan. menggunakan stored procedure untuk memproses pesanan dapat meningkatkan performa karena proses dilakukan langsung di server data base, tanpa bolak-balik query aplikasi.

BAB IV

IMPLEMENTASI

4.1 Tugas Praktikum

4.1.1 Soal 1

Note : Gunakan table UMKM pada Modul 2

1. Buatlah *stored procedure* AddUMKM yang menerima parameter nama_usaha (IN) dan jumlah_karyawan (IN) untuk menambahkan data UMKM baru ke dalam tabel umkm.
2. Tulis *stored procedure* UpdateKategoriUMKM yang menerima parameter id_kategori (IN) dan nama_baru (IN) untuk memperbarui nama kategori dalam tabel kategori_umkm berdasarkan ID kategori yang diberikan.
3. Buatlah *stored procedure* DeletePemilikUMKM yang menerima parameter id_pemilik (IN) untuk menghapus data pemilik UMKM berdasarkan ID yang diberikan dari tabel pemilik_umkm.
4. Tulis *stored procedure* AddProduk yang menerima parameter id_umkm (IN), nama_produk (IN), dan harga (IN) untuk menambahkan data produk ke dalam tabel produk_umkm.
5. Buatlah *stored procedure* GetUMKMByID yang menerima parameter id_umkm (IN) dan mengembalikan data UMKM yang sesuai dengan ID yang diberikan melalui parameter OUT.

4.1.2 Soal 2

Note : Gunakan table yang kalian gunakan pada Soal 2 Modul 1

1. Buatlah *stored procedure* UpdateDataMaster yang menerima parameter id (IN) dan nilai_baru (IN) untuk memperbarui nilai tertentu dalam tabel master di database Anda, dan mengembalikan status operasi melalui parameter OUT.
2. Buatlah *stored procedure* CountTransaksi yang tidak menerima parameter dan mengembalikan jumlah total entri yang terdaftar di tabel transaksi di database Anda melalui parameter OUT.

3. Buatlah *stored procedure* GetDataMasterByID yang menerima parameter id(IN) dan mengembalikan data dari tabel master terkait berdasarkan ID tersebut melalui parameter OUT.
4. Buatlah *stored procedure* UpdateFieldTransaksi yang menerima parameter id (IN), field1 (INOUT), dan field2 (INOUT) untuk memperbarui nilai dari dua kolom dalam tabel transaksi berdasarkan ID yang diberikan. Jika field1 atau field2 kosong, maka tetap gunakan nilai yang ada di database Anda.
5. Buatlah *stored procedure* DeleteEntriesByIDMaster yang menerima parameter id (IN) dan menghapus entri yang terkait berdasarkan ID tersebut dari tabel master di database Anda.

4.2 Source Code

4.2.1 Tugas Praktikum Soal No. 1

a) AddUMKM

```
DELIMITER //

CREATE PROCEDURE addUMKM (
    IN p_nama_usaha VARCHAR(200),
    IN p_jumlah_karyawan INT
)
BEGIN
    INSERT INTO umkm (nama_usaha,
jumlah_karyawan, tanggal_registrasi)
    VALUES (p_nama_usaha, p_jumlah_karyawan,
CURDATE());
END //

DELIMITER ;

CALL addUMKM('berusaha', 20);
SELECT * FROM umkm WHERE tanggal_registrasi =
CURDATE();
```

b) UpdateKategoriUMKM

```
DELIMITER //
```

```
CREATE PROCEDURE updateKategoriUMKM (  
    IN p_id_kategori INT,  
    IN p_nama_baru VARCHAR(100)  
)  
BEGIN  
    UPDATE kategori_umkm  
    SET nama_kategori = p_nama_baru  
    WHERE id_kategori = p_id_kategori;  
END //
```

```
DELIMITER ;
```

```
CALL updateKategoriUMKM(3, 'Kuliner');
```

c) DeletePemilikUMKM

```
DELIMITER //
```

```
CREATE OR REPLACE PROCEDURE deletePemilikUMKM (  
    IN p_id_pemilik INT  
)  
BEGIN  
    DELETE FROM pemilik_umkm WHERE id_pemilik =  
p_id_pemilik;  
END //
```

```
DELIMITER ;
```

```
CALL deletePemilikUMKM(7);
```

```
SELECT * FROM umkm;
```

```
SELECT * FROM pemilik_umkm;
```

d) AddProduk

```
DELIMITER //
```

```
CREATE PROCEDURE addProduk (  
    IN p_id_umkm INT,  
    IN p_nama_produk VARCHAR(200),  
    IN p_harga DECIMAL(15,2)  
)  
BEGIN  
    INSERT INTO produk_umkm (id_umkm,  
nama_produk, harga)  
VALUES (p_id_umkm, p_nama_produk, p_harga);  
END //
```

```
DELIMITER ;
```

```
CALL addProduk(2, 'Keripik Pisang', 15000.00);
```

e) getUMKMByID

```
DELIMITER //
```

```
CREATE PROCEDURE GetUMKMByID (  
    IN p_id_umkm INT,  
    OUT p_nama_usaha VARCHAR(200),  
    OUT p_jumlah_karyawan INT  
)  
BEGIN  
    SELECT nama_usaha, jumlah_karyawan  
    INTO p_nama_usaha, p_jumlah_karyawan  
    FROM umkm  
    WHERE id_umkm = p_id_umkm;  
END //
```

```
DELIMITER ;
```

```
CALL          GetUMKMByID(2,          @nama_usaha,
@jumlah_karyawan);
SELECT @nama_usaha, @jumlah_karyawan;
```

4.2.2 Tugas Praktikum Soal No. 2

a) UpdateDataMaster

```
DELIMITER //
```

```
CREATE PROCEDURE UpdateDataMaster (
    IN id INT,
    IN nilai_baru DECIMAL(10,2),
    OUT status_msg VARCHAR(100)
)
BEGIN
    DECLARE ROW_COUNT INT;

    UPDATE products SET harga = nilai_baru WHERE
id_produk = id;
    SET ROW_COUNT = ROW_COUNT();

    IF ROW_COUNT > 0 THEN
        SET status_msg = 'Update berhasil';
    ELSE
        SET status_msg = 'Data tidak ditemukan';
    END IF;
END //
```

```
DELIMITER ;
```

```
CALL          UpdateDataMaster(101,          250000.00,
@status_msg);
SELECT @status_msg;
```

b) CountTransaksi

```
DELIMITER //
```

```
CREATE PROCEDURE CountTransaksi (  
    OUT total_transaksi INT  
)  
BEGIN  
    SELECT COUNT(*) INTO total_transaksi FROM  
orders;  
END //
```

```
DELIMITER ;  
Call CountTransaksi(@total_transaksi);  
SELECT @total_transaksi;
```

c) GetDataMasterByID

```
DELIMITER //
```

```
CREATE PROCEDURE GetDataMasterByID (  
    IN id INT,  
    OUT nama_produk VARCHAR(100),  
    OUT harga_produk DECIMAL(10,2)  
)  
BEGIN  
    SELECT NAME, harga INTO nama_produk,  
harga_produk  
    FROM products  
    WHERE id_produk = id;  
END //
```

```
DELIMITER ;
```

```
CALL      GetDataMasterByID(2,      @nama_produk,  
@harga_produk);  
SELECT @nama_produk, @harga_produk;
```

d) UpdateFieldTransaksi

```
DELIMITER //  
CREATE PROCEDURE UpdateFieldTransaksi (  
    IN id INT,  
    INOUT total_price_param DECIMAL(10,2),  
    INOUT      status_param      ENUM('Pending',  
'Shipped', 'Completed', 'Canceled')  
)  
BEGIN  
    DECLARE current_price DECIMAL(10,2);  
    DECLARE      current_status      ENUM('Pending',  
'Shipped', 'Completed', 'Canceled');  
    SELECT      total_price,      STATUS      INTO  
current_price, current_status  
    FROM orders  
    WHERE order_id = id;  
    IF total_price_param IS NULL THEN  
        SET total_price_param = current_price;  
    END IF;  
    IF status_param IS NULL THEN  
        SET status_param = current_status;  
    END IF;  
    UPDATE orders  
    SET total_price = total_price_param,  
        STATUS = status_param  
    WHERE order_id = id;  
END //  
DELIMITER ;
```



```

SET @total_price = NULL;
SET @status = NULL;
CALL      UpdateFieldTransaksi(3,      @total_price,
@status);

SET @total_price = 175000.00;
SET @status = 'Shipped';

CALL      UpdateFieldTransaksi(3,      @total_price,
@status);

```

e) DeleteEntriesByIDMaster

```

DELIMITER //

CREATE PROCEDURE DeleteEntriesByIDMaster (
    IN id INT
)
BEGIN
    DELETE FROM products WHERE id_produk = id;
END //

DELIMITER ;

CALL DeleteEntriesByIDMaster(7);

```

4.3 Hasil

4.3.1 Tugas Praktikum Soal No. 1

- AddUMKM

```











17 CALL addUMKM('berusaha', 20);
18 SELECT nama_usaha, jumlah_karyawan, tanggal_registrasi FROM umkm WHERE tanggal_registrasi = CURDATE();
19

```

nama_usaha	jumlah_karyawan	tanggal_registrasi
berusaha	20	2025-04-22

- UpdateKategoriUMKM

```
35 CALL updateKategoriUMKM(3, 'Kuliner');
36 SELECT id_kategori, nama_kategori FROM kategori_umkm WHERE id_kategori = 3;
37
```

				
 (Read Only)    				
<input type="checkbox"/>	id_kategori	nama_kategori		
<input type="checkbox"/>	3	Kuliner		

- DeletePemilikUMKM

id_umkm	nama_usaha	id_pemilik	id_kategori	id_skala	id_kabupaten_kota	alamat_usaha	nib
1	Warung Sambil Hejo	1	1	1	1	Jl. Pahlawan No. 123, Ban...	29B 912000123456
2	Butik Batik Pesona	2	2	2	2	Jl. Merdeka No. 45, Bekasi	26B 912000123456
3	Kerajinan Anyaman Rahayu	3	3	1	1	Jl. Sukajadi No. 78, Bandung	28B 912000123456
4	Taman Buah Sejahtera	4	4	2	3	Jl. Veteran No. 56, Bogor	25B 912000123457
5	Ternak Ayam Makmur	(NULL)	5	1	12	Jl. Cendana No. 34, Bekasi	26B 912000123457
6	Budidaya Ikan Barokah	6	6	2	4	Jl. Kemuning No. 67, Cimahi	27B 912000123457
7	Bengkel Las Abadi	(NULL)	7	1	10	Jl. Teratai No. 89, Bandung	27B 912000123457
8	Toko Sembako Barokah	8	8	2	17	Jl. Cikutra No. 23, Garut	25B 912000123457
9	Furniture Kayu Jati	9	9	3	15	Jl. Sudirman No. 56, Cianjur	28B 912000123457

id_pemilik	nik	nama_lengkap	jenis_kelamin	alamat	nomor_telepon	email
1	3273012505780001	Ahmad Sudrajat	Laki-laki	Jl. Pahlawan No. 123, Ban...	29B 081234567890	ahmad.sudrajat@gmail.com
2	3217016004850002	Siti Rahayu	Perempuan	Jl. Merdeka No. 45, Bekasi	26B 085678901234	siti.rahayu@gmail.com
3	3273025601900003	Budi Santoso	Laki-laki	Jl. Sukajadi No. 78, Bandung	28B 081345678901	budi.santoso@gmail.com
4	3271046502870004	Dewi Lestari	Perempuan	Jl. Veteran No. 56, Bogor	25B 087890123456	dewi.lestari@gmail.com
6	3277054408920006	Rina Anggraini	Perempuan	Jl. Kemuning No. 67, Cimahi	27B 082345678901	rina.anggraini@gmail.com
8	3215026302860008	Ani Yudhoyono	Perempuan	Jl. Cikutra No. 23, Garut	25B 083567890123	ani.yudhoyono@gmail.com
9	3601014507830009	Hendra Wijaya	Laki-laki	Jl. Sudirman No. 56, Cianjur	28B 085678901234	hendra.wijaya@gmail.com
10	3216028308910010	Maya Sari	Perempuan	Jl. Gatot Subroto No. 78, ...	33B 087890123456	maya.sari@gmail.com
11	3214013011820011	Rudi Hartono	Laki-laki	Jl. Setiabudi No. 90, Kun...	30B 089012345678	rudi.hartono@gmail.com
12	3279027105860012	Tina Marlina	Perempuan	Jl. Pasteur No. 45, Matal...	30B 081234567890	tina.marlina@gmail.com

- AddProduk

id_umkm	nama_produk	harga
2	Batik Tulis Mega Mendung	750000.00
2	Batik Cap Kujang	350000.00
2	Kemeja Batik Pria	275000.00
2	Dress Batik Modern	325000.00
2	Keripik Pisang	15000.00

- getUMKMByID

```
CALL GetUMKMByID(6, @nama_usaha, @jumlah_karyawan);
SELECT @nama_usaha, @jumlah_karyawan;
```

1 Result

2 Profiler

3 Messages

4 Table Data

(Read Only)

@nama_usaha	@jumlah_karyawan
Budidaya Ikan Barokah	21B 8

4.3.2 Tugas Praktikum Soal No. 2

- UpdateDataMaster

@status_msg				
Update berhasil			15B	

id_produk	name	kategori	harga	stok
1	Sharingan Bait	Fishing Lures	250000.00	10
2	Explosion Bait	Fishing Lures	77.77	20
3	Ackerman Hook	Hooks	129.99	15
4	Zura Rod	Fishing Rods	250000.00	25
5	Divine Water Bait	Fishing Lures	55.55	30
6	Lost Navigator Compass	Accessories	10.10	50
7	100 Billion Percent Rod	Fishing Rods	250000.00	5
8	Elegant Hook	Hooks	49.49	35
9	Love Is War Net	Nets	250000.00	12

- CountTransaksi

```
--
126 CREATE PROCEDURE CountTransaksi (
127     OUT total_transaksi INT
128 )
129 BEGIN
130     SELECT COUNT(*) INTO total_transaksi FROM orders;
131 END //
132
133 DELIMITER ;
134 CALL CountTransaksi(@total_transaksi);
135 SELECT @total_transaksi;
136
```

1 Result	2 Profiler	3 Messages	4 Table Data	5 Info
(Read Only)				
@total_transaksi				
10				

- GetDataMasterByID

```
-- GetDataMasterByID
DELIMITER //

CREATE PROCEDURE GetDataMasterByID (
    IN id INT,
    OUT nama_produk VARCHAR(100),
    OUT harga_produk DECIMAL(10,2)
)
BEGIN
    SELECT NAME, harga INTO nama_produk, harga_produk
    FROM products
    WHERE id_produk = id;
END //

DELIMITER ;
CALL GetDataMasterByID(2, @nama_produk, @harga_produk);
SELECT @nama_produk, @harga_produk;
```

1 Result	2 Profiler	3 Messages	4 Table Data	5 Info
(Read Only)				
@nama_produk @harga_produk				
Explosion Bait 14B 77.77				

- UpdateFieldTransaksi

```

192 SELECT * FROM orders;
193
194 SET @total_price = 175000.00;
195 SET @status = 'Shipped';
196
197 CALL UpdateFieldTransaksi(3, @total_price, @status);
198
199

```

order_id	customer_id	order_date	total_price	status	keterangan
1	1	2025-03-31 19:03:45	99.99	Pending	(NULL)
2	2	2025-03-31 19:03:45	77.77	Shipped	pesanan sedang dalam tujuan
3	3	2025-03-31 19:03:45	175000.00	Shipped	pesanan sudah samapi di alamat tujuan
4	4	2025-03-31 19:03:45	88.88	Canceled	(NULL)
5	5	2025-03-31 19:03:45	55.55	Pending	(NULL)
6	6	2025-03-31 19:03:45	10.10	Completed	pesanan sudah samapi di alamat tujuan
7	7	2025-03-31 19:03:45	150.00	Pending	(NULL)
8	8	2025-03-31 19:03:45	49.49	Shipped	pesanan sedang dalam tujuan
9	9	2025-03-31 19:03:45	120.00	Pending	(NULL)

- DeleteEntriesByIDMaster

```

200 -- DeleteEntriesByIDMaster
201 DELIMITER //
202
203 CREATE PROCEDURE DeleteEntriesByIDMaster (
204     IN id INT
205 )
206 BEGIN
207     DELETE FROM products WHERE id_produk = id;
208 END //
209
210 DELIMITER ;
211
212 CALL DeleteEntriesByIDMaster(7);
213 SELECT * FROM products;
214
215

```

id_produk	name	kategori	harga	stok	deskripsi
1	Sharingan Bait	Fishing Lures	250000.00	10	Menarik perhatian ikan dengan Genjutsu!
2	Explosion Bait	Fishing Lures	77.77	20	Bait ini hanya bisa digunakan sekali. EXPLOSION!
3	Ackerman Hook	Hooks	129.99	15	Secepat refleks Levi!
4	Zura Rod	Fishing Rods	250000.00	25	Bukan Katsura, tapi pancing Zura!
5	Divine Water Bait	Fishing Lures	55.55	30	Diberkahi oleh Aqua, tapi jangan percaya airnya.
6	Lost Navigator Compass	Accessories	10.10	50	Dijamin bikin kamu kehilangan arah seperti Zoro.
8	Elegant Hook	Hooks	49.49	35	Bait ini memiliki ELEGANCE, cocok untuk Anya.
9	Love Is War Net	Nets	250000.00	12	Strategi cerdas menangkap ikan, tapi hati-hati jangan jat..

4.4 Penjelasan

4.4.1 Tugas Praktikum Soal No. 1

a. addUMKM

Prosedur ini digunakan untuk menambahkan data UMKM baru ke dalam tabel umkm. Prosedur menerima dua parameter *input*:

- p_nama_usaha: nama usaha (VARCHAR)
- p_jumlah_karyawan: jumlah karyawan (INT) Ketika prosedur dipanggil (CALL addUMKM('berusaha', 20);), maka akan disisipkan data baru ke tabel umkm dengan tanggal registrasi diisi otomatis menggunakan CURDATE() (tanggal saat ini). Setelah itu, dilakukan seleksi data untuk menampilkan UMKM yang baru saja ditambahkan berdasarkan tanggal registrasi hari ini.

b. updateKategoriUMKM

Prosedur ini digunakan untuk memperbarui nama kategori UMKM. Prosedur menerima dua parameter:

- p_id_kategori: ID kategori yang ingin diubah
- p_nama_baru: nama kategori baru Prosedur akan melakukan UPDATE pada tabel kategori_umkm berdasarkan id_kategori. Setelah dipanggil (CALL updateKategoriUMKM(3, 'Kuliner');), dilakukan seleksi data untuk menampilkan hasil perubahan.

c. deletePemilikUMKM

Prosedur ini digunakan untuk menghapus data pemilik UMKM berdasarkan ID. Prosedur menerima:

- p_id_pemilik: ID pemilik yang akan dihapus Saat prosedur dijalankan (CALL deletePemilikUMKM(7);), maka data pada tabel pemilik_umkm dengan ID tersebut akan dihapus. Setelah itu, dua query SELECT digunakan untuk melihat isi terkini dari tabel umkm dan pemilik_umkm.

d. addProduk

Prosedur ini digunakan untuk menambahkan produk baru milik UMKM. Parameter yang diterima:

- p_id_umkm: ID dari UMKM yang memiliki produk tersebut
- p_nama_produk: nama produk
- p_harga: harga produk Data akan ditambahkan ke tabel produk_umkm. Setelah menjalankan prosedur (CALL addProduk(2, 'Keripik Pisang', 15000.00);), dilakukan seleksi untuk menampilkan produk-produk milik UMKM dengan id_umkm = 2.

e. GetUMKMByID

Prosedur ini digunakan untuk mengambil informasi UMKM berdasarkan ID. Parameter:

- p_id_umkm: ID UMKM yang dicari
- p_nama_usaha dan p_jumlah_karyawan: parameter output untuk menampung hasil query Data nama usaha dan jumlah karyawan dari umkm dengan ID tertentu akan disimpan ke dalam variabel output. Setelah menjalankan prosedur (CALL GetUMKMByID(6, @nama_usaha, @jumlah_karyawan);), nilai-nilai tersebut ditampilkan dengan SELECT @nama_usaha, @jumlah_karyawan.

4.4.2 Tugas Praktikum Soal No. 2

Prosedur pertama adalah `UpdateDataMaster`, yang digunakan untuk memperbarui harga produk pada tabel `products` berdasarkan `id_produk`. Prosedur ini menerima dua parameter *input* (`id` dan `nilai_baru`) serta satu parameter output (`status_msg`) untuk memberikan pesan status setelah eksekusi. Jika baris berhasil diperbarui, maka `status_msg` di-set menjadi "Update berhasil"; jika tidak ditemukan data, maka akan berisi "Data tidak ditemukan".

Selanjutnya adalah `CountTransaksi`, yang berfungsi untuk menghitung jumlah total transaksi yang ada di tabel `orders`. Prosedur ini hanya memiliki satu parameter output yaitu `total_transaksi`, yang akan menyimpan hasil dari perhitungan `COUNT(*)`. Prosedur ini berguna untuk menampilkan jumlah keseluruhan order yang sudah tercatat dalam sistem.

Kemudian terdapat prosedur `GetDataMasterByID`, yang memungkinkan pengguna untuk mengambil data produk berdasarkan ID produk tertentu. Prosedur ini menerima parameter *input* `id`, serta dua parameter output: `nama_produk` dan `harga_produk`. Nilai-nilai ini akan diisi dari hasil seleksi kolom `NAME` dan `harga` dari tabel `products`.

Prosedur berikutnya adalah `UpdateFieldTransaksi`, yang lebih kompleks karena menggunakan parameter `INOUT`. Prosedur ini digunakan untuk memperbarui dua field dalam tabel `orders`, yaitu `total_price` dan `status`, berdasarkan `order_id`. Jika nilai `INOUT` tersebut dikirim sebagai `NULL`, maka prosedur akan mengambil nilai sebelumnya dari tabel untuk dijadikan pengganti. Hal ini memungkinkan fleksibilitas dalam mengupdate salah satu atau kedua nilai sekaligus.

Prosedur terakhir adalah `DeleteEntriesByIDMaster`, yang berfungsi untuk menghapus data produk dari tabel `products` berdasarkan `id_produk`. Prosedur ini sederhana, hanya memiliki satu parameter *input* (`id`), dan akan menghapus data tanpa mengembalikan hasil selain efek perubahan data.

BAB V

PENUTUP

5.1 Analisa

Pada praktikum ini, kami belajar bagaimana membuat *stored procedure* dalam basis data. *Stored procedure* adalah sekumpulan perintah SQL yang disimpan di dalam database dan dapat dijalankan secara berulang-ulang dengan cukup memanggil nama prosedurnya. Konsep ini sangat berguna karena memungkinkan kita untuk menyimpan logika bisnis secara langsung di sisi server, sehingga tidak perlu menulis ulang query yang kompleks setiap kali dibutuhkan.

Contohnya, prosedur `addUMKM` digunakan untuk menambahkan data UMKM baru dengan parameter nama usaha dan jumlah karyawan, serta otomatis mencatat tanggal registrasi. Dengan prosedur ini, proses insert data menjadi lebih ringkas dan konsisten. Begitu juga dengan prosedur `updateKategoriUMKM` yang mempermudah proses update nama kategori UMKM tanpa harus menulis query UPDATE secara manual tiap kali ada perubahan.

Selain itu, ada juga prosedur yang memberikan logika tambahan, seperti `UpdateDataMaster` yang mengubah harga produk dan sekaligus memberikan pesan status apakah proses update berhasil atau tidak. Prosedur ini tidak hanya melakukan perubahan data, tetapi juga menyertakan mekanisme umpan balik melalui parameter output, yang sangat berguna dalam pengembangan aplikasi.

Beberapa prosedur juga dirancang untuk pengambilan data, seperti `GetDataMasterByID` atau `GetUMKMByID`, yang memungkinkan aplikasi untuk langsung mendapatkan informasi penting berdasarkan ID, tanpa perlu menulis query seleksi yang panjang. Bahkan ada prosedur seperti `CountTransaksi` yang melakukan agregasi data untuk menghitung jumlah transaksi, atau `UpdateFieldTransaksi` yang memperbarui dua kolom sekaligus berdasarkan kondisi tertentu, dan memberikan fleksibilitas dengan parameter INOUT.

Dari praktikum ini, kami jadi lebih memahami bahwa *stored procedure* sangat membantu dalam mengelola data secara lebih terstruktur, efisien, dan aman. Selain mengurangi penulisan kode SQL berulang, *stored procedure* juga menjaga konsistensi, meningkatkan performa karena dijalankan langsung oleh server

database, dan dapat dikontrol aksesnya sehingga membantu dalam aspek keamanan data.

Secara keseluruhan, penggunaan *stored procedure* adalah pendekatan yang sangat efektif dalam pengembangan sistem basis data, karena menyatukan logika bisnis dan manipulasi data dalam satu wadah yang terkelola dengan baik, serta sangat mendukung kebutuhan operasional dan pengembangan aplikasi ke depannya.

5.2 Kesimpulan

Berdasarkan praktikum yang telah dilakukan, dapat disimpulkan bahwa *stored procedure* merupakan fitur penting dalam sistem manajemen basis data yang berfungsi untuk menyimpan dan menjalankan sekumpulan perintah SQL secara terstruktur. Dengan menggunakan *stored procedure*, kita dapat mengotomatisasi proses manipulasi data seperti penambahan, pembaruan, penghapusan, dan pengambilan data tanpa harus menulis ulang query SQL setiap kali proses tersebut dibutuhkan. *Stored procedure* tidak hanya membuat pengelolaan data menjadi lebih efisien, tetapi juga membantu menjaga konsistensi dan keamanan dalam pengembangan sistem berbasis data.

Beberapa hal penting yang dapat disimpulkan dari praktikum ini antara lain:

- a. *Stored procedure* mempermudah pelaksanaan operasi berulang seperti insert, update, delete, dan select.
- b. *Stored procedure* memungkinkan penggunaan parameter *input*, *output*, maupun *input-output* untuk fleksibilitas data yang tinggi.
- c. *Stored procedure* membantu menjaga konsistensi dan validasi logika bisnis langsung di level database.
- d. *Stored procedure* meningkatkan efisiensi dan performa sistem karena dieksekusi langsung oleh server database.
- e. *Stored procedure* dapat memperkuat keamanan data dengan membatasi akses hanya melalui prosedur tertentu, bukan langsung ke tabel.
- f. Praktikum ini juga memperkuat pemahaman tentang penggunaan kondisi logika, manipulasi data dinamis, dan interaksi antar tabel dalam SQL.