MODUL PRAKTIKUM

PEMROGRAMAN BASIS DATA

PRODI INFORMATIKA

UNIVERSITAS AMIKOM YOGYAKARTA

2022

PENGESAHAN

NAMA DAN KODE MATA KULIAH	: *PEMROGRAMAN BASIS DATA (ST116)*
Capaian Kompetensi	:
Tata Tertib Praktikum	:
Tim Penyusun	: Irwan Oyong, M.Kom Juarisman, M.Kom Novi Prisma Yunita, M.Kom
Tim Penyunting	: -

Disahkan oleh

Koordinator Praktikum	Ketua Program Studi
_____	Windha Mega Pradnya Dhuhita, M.Kom.

PENGANTAR

Puji syukur kami haturkan kehadirat Tuhan Yang Maha Esa, berkat anugerah berupa kesehatan, kemampuan, dan waktu luang sehingga kami dapat menyelesaikan modul praktikum mata kuliah Pemrograman Basis Data tepat waktu, dan tanpa halangan yang berarti.

Modul praktikum Pemrograman Basis Data ini terdiri dari materi dan langkah-langkah praktikum yang akan dibahas selama mata kuliah Pemrograman Basis data, modul praktikum ini harapannya dapat dijadikan acuan oleh dosen dan mahasiswa dalam proses pembelajaran di kelas teori maupun praktikum. Modul ini terdiri dari materi *advanced* dari SQL yang *compatible* untuk *database tools* SQL Server dan MySQL, dan dilengkapi dengan lebih dari satu studi kasus yang harapannya dapat memberikan *insight* terkait penggunaan basis data di dunia nyata.

Semoga dengan adanya modul praktikum ini dapat memudahkan dosen dan mahasiswa dalam hal penyampaian dan penerimaan materi mata kuliah Pemrograman Basis Data, sehingga dapat memudahkan pula dalam meraih capaian pembelajaran yang diharapkan.

Tim Penyusun.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I FASE PEMROSESAN QUERY DI SQL

1.1 Tujuan

Setelah praktikum ini usai, mahasiswa diharapkan:

- mampu memahami fase-fase pemrosesan *query* di SQL
- mampu menjelaskan urutan pemrosesan *query* di SQL
- mampu menjelaskan perbedaan yang ada terkait fase pemrosesan *query* di aplikasi basisdata berbasis SQL

1.2 Dasar Teori

QUERY PROCCESING

Query processing atau pemrosesan *query* adalah sebuah proses di mana deklarasi *query* diterjemahkan menjadi operasi manipulasi data pada level rendah. Tujuan pemrosesan *query* adalah untuk mengubah *query* yang ditulis dalam bahasa tingkat tinggi ke dalam bahasa tingkat rendah. Pernyataan utama yang digunakan untuk mengambil data dalam T-SQL adalah pernyataan SELECT. Berikut ini adalah klausa utama pada sebuah *query*:

1. SELECT
2. FROM
3. WHERE
4. GROUP BY
5. HAVING
6. ORDER BY

Meskipun dituliskan dengan bahasa yang mudah dipahami, sebenarnya fase pemrosesan *query* tidak sama dengan bagaimana sebuah *query* dituliskan pada klausa SELECT. Pemrosesan *query* di SQL dimulai dari klausa FROM, dan bukan SELECT. Berikut ini adalah urutan dari sebuah *query processing*:

1. FROM
2. WHERE
3. GROUP BY

4. HAVING
5. SELECT
6. ORDER BY

Pada setiap tahap bertindak pada satu tabel atau lebih dimana sebagai *input* dan *output* yang merupakan tabel virtual. Tabel *output* dari satu tahap dianggap sebagai *input* untuk tahap berikutnya. Hal ini sesuai dengan operasi pada relasi yang menghasilkan relasi.

Perhatikan SELECT statement di bawah ini:

```
SELECT country, YEAR(hiredate) AS yearhired, COUNT(*) AS numemployees
FROM HR.Employees
WHERE hiredate >= '20030101'
GROUP BY country, YEAR(hiredate)
HAVING COUNT(*) > 1
ORDER BY country , yearhired DESC;
```

SELECT statement di atas akan menampilkan data dari tabel employees, hanya karyawan yang sudah bekerja setelah tahun 2003 saja yang akan ditampilkan. Untuk setiap grup yang memenuhi syarat, SELECT statement akan menampilkan *record* tahun perekrutan dan jumlah karyawan, diurutkan berdasarkan negara dan tahun perekrutan, dalam urutan menurun.

Paparan berikut ini memberikan deskripsi singkat tentang apa yang terjadi di setiap tahap *query processing*:

FROM

Pada tahap pertama, klausa FROM dievaluasi. Pada tahap ini, ditentukan data apa saja yang akan ditampilkan, apakah menampilkan data dari satu tabel saja atau gabungan tabel lain yang memenuhi syarat.

```
SELECT country, YEAR(hiredate) AS yearhired, COUNT(*) AS numemployees
FROM HR.Employees
```

```
WHERE hiredate >= '20030101'

GROUP BY country, YEAR(hiredate)

HAVING COUNT(*) > 1

ORDER BY country, yearhired DESC;
```

Kemudian, *output* dari tahap ini adalah hasil tabel dengan semua baris dari tabel *input*, dalam hal ini semua baris dan kolom yang ada di tabel employees, yakni 3 buah kolom: empid, hiredate, dan country; dan 9 baris data seperti tertera pada gambar di bawah ini.

empid	hiredate	country
1	2002-05-01	USA
2	2002-08-14	USA
3	2002-04-01	USA
4	2003-05-03	USA
5	2003-10-17	UK
6	2003-10-17	UK
7	2004-01-02	UK
8	2004-03-05	USA
9	2004-11-15	UK

Gambar 1. 1 *Output* evaluasi klausa FROM

Output berupa *result-set* dari evaluasi klausa FORM di atas, kemudian menjadi *input* dari evaluasi klausa selanjutnya.

WHERE

Tahap kedua pemrosesan *query* adalah evaluasi klausa WHERE. Klausa WHERE memfilter baris berdasarkan kriteria tertentu, hanya baris yang memenuhi kriteria saja yang akan ditampilkan.

```
SELECT country, YEAR(hiredate) AS yearhired, COUNT(*) AS numemployees

FROM HR.Employees

WHERE hiredate >= '20030101'

GROUP BY country, YEAR(hiredate)

HAVING COUNT(*) > 1

ORDER BY country , yearhired DESC;
```

Berdasarkan SELECT statement di atas, tahap pemfilteran WHERE hanya memfilter baris untuk karyawan yang dipekerjakan pada atau setelah 1 Januari 2003. Pada tahap ini, enam *record* data akan ditampilkan. Selanjutnya *output* pada tahap ini adalah *input* di tahap selanjutnya, yakni evaluasi klausa GROUP BY.

empid	hiredate	country
4	2003-05-03	USA
5	2003-10-17	UK
6	2003-10-17	UK
7	2004-01-02	UK
8	2004-03-05	USA
9	2004-11-15	UK

Gambar 1. 2 *Output* evaluasi kluasa WHERE

Kesalahan yang biasa dilakukan pada tahap ini adalah mencoba merujuk klausa WHERE dengan menggunakan alias kolom yang didefinisikan pada klausa SELECT. Ini tidak diperbolehkan karena klausa WHERE dievaluasi sebelum klausa SELECT. Sebagai contoh, perhatikan SELECT statement berikut ini:

```
SELECT country, YEAR(hiredate) AS yearhired
FROM HR.Employees
WHERE yearhired >= 2003; ← yearhired is invalid
```

Query ini gagal dengan pesan *error* berikut:

```
Msg 207, Level 16, State 1, Line 3
Invalid column name 'yearhired'.
```

`Yearhired` tidak dikenali dan dianggap tidak valid oleh *database engine* karena pada dasarnya klausa WHERE dievaluasi terlebih dahulu sebelum klausa SELECT. Sebagai alternatif, pada klausa WHERE cukup menggunakan nama kolom aslinya saja, tidak menggunakan alias. Ganti baris berikut: `WHERE yearhired >= 2003;` menjadi `WHERE YEAR(hiredate) >= 2003;`

KLAUSA GROUP BY

Fase pemrosesan *query* setelah WHERE adalah GROUP BY. Klausa GROUP BY digunakan untuk mengelompokkan data sejenis sesuai dengan kriteria kelompok yang didefinisikan pada klausa ini.

```
SELECT country, YEAR(hiredate) AS yearhired, COUNT(*) AS numemployees
FROM HR.Employees
WHERE hiredate >= '20030101'
GROUP BY country, YEAR(hiredate)
HAVING COUNT(*) > 1
ORDER BY country , yearhired DESC;
```

Berdasarkan SELECT statement di atas, baris data akan dikelompokkan berdasarkan kolom `country`, dan tahun `hiredate`. Enam buah baris data pada fase sebelumnya akan dievaluasi dan dikelompokkan. Hasil pengelompokkannya adalah sebagaimana gambar di bawah ini.

group country	group YEAR(hiredate)	detail empid	detail country	detail hiredate
UK	2003	5	UK	2003-10-17
		6	UK	2003-10-17
UK	2004	7	UK	2004-01-02
		9	UK	2004-11-15
USA	2003	4	USA	2003-05-03
USA	2004	8	USA	2004-03-05

Gambar 1. 3 *Output* dari evaluasi klausa GROUP BY

Tampak pada gambar di atas, grup UK tahun 2003 terdapat dua baris detail terkait dengan karyawan 5 dan 6; grup untuk UK tahun 2004 juga terdapat dua baris detail terkait dengan karyawan 7 dan 9; grup untuk Amerika Serikat, 2003 terdapat satu baris detail terkait dengan karyawan 4; grup untuk Amerika Serikat, 2004 juga terdapat satu baris detail terkait dengan karyawan 8.

Hasil akhir SELECT statement ini memiliki satu baris yang mewakili setiap grup (kecuali difilter). Oleh karena itu, ekspresi di semua tahap yang terjadi setelah tahap pengelompokan saat ini terbatas. Semua baris yang diproses pada tahap berikutnya harus menjamin satu nilai per grup. Jika merujuk ke elemen dari daftar GROUP BY (misalnya country) sudah bisa dilakukan, sehingga referensi seperti itu diperbolehkan. Namun, jika ingin merujuk ke elemen yang bukan

bagian dari daftar GROUP BY misalnya: empid, elemen tersebut harus terkandung dalam fungsi gregasi seperti MAX atau SUM.

KLAUSA HAVING

Tahapan pemrosesan *query* selanjutnya adalah evaluasi klausa HAVING.

```
SELECT country, YEAR(hiredate) AS yearhired, COUNT(*) AS numemployees
FROM HR.Employees
WHERE hiredate >= '20030101'
GROUP BY country, YEAR(hiredate)
HAVING COUNT(*) > 1
ORDER BY country , yearhired DESC;
```

Dalam hal ini, klausa HAVING menggunakan predikat COUNT(*) > 1, yang berarti hanya grup negara dan tahun perekrutan yang memiliki lebih dari satu karyawan saja yang akan ditampilkan. Jika melihat jumlah baris yang terkait dengan setiap grup pada langkah sebelumnya, akan terlihat bahwa hanya grup UK tahun 2003 dan UK tahun 2004 yang memenuhi syarat. Evaluasi klausa HAVING menghasilkan nilai *true*, *false*, atau *unknown*. Hanya kelompok yang bernilai *true* yang ditampilkan pada tahap ini. Gambar di bawah ini adalah *output* dari evaluasi klausa HAVING.

group country	group YEAR(hiredate)	detail empid	detail country	detail hiredate
UK	2003	5	UK	2003-10-17
		6	UK	2003-10-17
UK	2004	7	UK	2004-01-02
		9	UK	2004-11-15

Gambar 1. 4 *Output* dari evaluasi klausa HAVING

SELECT

Tahapan *query processing* selanjutnya adalah yang bertanggung jawab untuk memproses klausa SELECT. Langkah pertama adalah mengevaluasi ekspresi dalam daftar SELECT dan menghasilkan atribut hasil. Ini termasuk menetapkan atribut dengan nama jika berasal dari ekspresi. Ingatlah bahwa jika SELECT statement menggunakan dikelompokkan, setiap grup diwakili oleh satu baris dalam hasilnya. Dalam SELECT statement, dua grup tetap ada setelah pemrosesan filter HAVING. Oleh karena itu, langkah ini menghasilkan dua baris. Dalam hal ini, daftar SELECT ini untuk setiap country dan tahun pesanan mengelompokkan baris dengan atribut berikut: `country`, `YEAR(hiredate)` alias sebagai `yearhired`, dan `COUNT(*)` alias sebagai `numemployees`.

Langkah kedua dalam tahap ini berlaku jika menunjukkan klausa DISTINCT, dalam hal ini langkah ini menghapus nilai duplikat. Gambar di bawah ini adalah hasil evaluasi dari klausa SELECT.

```
country  yearhired  numemployees
-------- ---------- ------------
UK       2003       2
UK       2004       2
```

Gambar 1. 5 *Output* evaluasi klausa SELECT

Tahap kelima mengembalikan hasil *record*. Oleh karena itu, urutan baris tidak ditentukan. Dalam kasus SELECT statement ini, ada klausa ORDER BY yang menjamin urutan dalam hasil *record*. Yang penting untuk dicatat adalah bahwa hasil dari tahap yang memproses klausa SELECT masih bersifat relasional.

Pada tahap ini menetapkan alias kolom ditentukan, seperti *yearhired* dan *numemployees*. Artinya bahwa alias kolom yang baru dibuat tidak dikenali oleh klausa yang diproses pada tahap sebelumnya, seperti FROM, WHERE, GROUP BY, dan HAVING. Bahkan, alias yang dibuat pada klausa SELECT juga tidak dikenali tidak terlihat oleh ekspresi lain yang muncul dalam daftar SELECT yang sama.

```
SELECT empid, YEAR(hiredate) AS yearhired, yearhired - 1 AS prevyear
FROM employees;
```

Query di atas gagal dengan pesan *error* berikut:

```
Msg 207, Level 16, State 1, Line 1
Invalid column name 'yearhired'.
```

Alasan mengapa ini tidak diperbolehkan adalah bahwa secara konseptual SELECT statement mengevaluasi semua ekspresi yang muncul dalam tahap pemrosesan SELECT statement yang sama dengan cara sekaligus. Perhatikan penggunaan kata secara konseptual. SQL Server tidak akan selalu memproses semua ekspresi secara fisik pada titik waktu yang sama, tetapi harus menghasilkan hasil seolah-olah itu terjadi. Perilaku ini berbeda dari banyak bahasa pemrograman lainnya di mana ekspresi biasanya dievaluasi dalam urutan kiri-ke-kanan, membuat hasil yang dihasilkan dalam satu ekspresi terlihat oleh yang muncul di sebelah kanannya.

KLAUSA ORDER BY

Tahap ke-enam berlaku jika SELECT statement dilengkapi dengan klausa ORDER BY. Tahap ini mengembalikan *result-set* dengan urutan tertentu sesuai dengan ekspresi yang muncul dalam daftar ORDER BY.

```
SELECT country, YEAR(hiredate) AS yearhired, COUNT(*) AS numemployees

FROM HR.Employees

WHERE hiredate >= '20030101'

GROUP BY country, YEAR(hiredate)

HAVING COUNT(*) > 1

ORDER BY country , yearhired DESC;
```

Berdasarkan baris ORDER BY di atas, *result-set* akan diurutkan berdasarkan `country` dengan urutan A-Z atau *ascending*, kemudian berdasarkan `yearhired` secara *descending*.

```
country  yearhired  numemployees
-------- ---------- ------------
UK       2004       2
UK       2003       2
```

Gambar 1. 6 *Output* dari evaluasi klausa ORDER BY

Perhatikan bahwa klausa ORDER BY adalah klausa pertama dan satu-satunya yang diizinkan untuk merujuk ke alias kolom yang ditentukan dalam klausa SELECT. Itu karena klausa ORDER BY adalah satu-satunya yang dievaluasi setelah klausa SELECT.

Catatan: *logical query processing phase* di atas berlaku untuk SQL Server, dan hanya sebagian saja yang berlaku untuk MySQL.

1.3 Alat dan Bahan

Database tools	MySQL
File import	Foodscenter.sql

1.4 Langkah Percobaan

Sebelum praktikum dimulai, buatlah basisdata dengan nama foodscenters lalu tambahkan 4 digit NIM anda, contoh: foodscenter_3456. Kemudian import file foofscenter.sql ke dalam basisdata anda.

Lakukan percobaan-percobaan berikut ini:

1. Analisalah SELECT statement di bawah ini. Paparkan alasan kenapa hasil eksekusi SELECT statement berikut ini mengembalikan pesan error!

```
SELECT id_menu,jenis AS tipe, nama, harga
FROM menu
WHERE harga >= 12000
GROUP BY tipe;
```

2. Perbaiki SELECT statement di atas sehingga dapat dieksekusi dan menghasilkan *result-set* yang diinginkan.
3. Analisa SELECT statement di bawah ini. Paparkan alasan kenapa hasil eksekusi SELECT statement berikut ini mengembalikan pesan error!

```
SELECT id_detail, id_transaksi, id_menu, qty as pesanan, harga
FROM detail_transaksi
GROUP BY id_detail, id_transaksi,id_menu, pesanan, harga
HAVING(pesanan >= 2);
```

4. Perbaiki SELECT statement di atas sehingga dapat dieksekusi dan menghasilkan *result-set* yang diinginkan.
5. Analisa SELECT statement di bawah ini. Paparkan alasan kenapa hasil eksekusi SELECT statement berikut ini mengembalikan pesan error!

```
SELECT id_detail, id_transaksi, id_menu, qty as pesanan, harga
FROM detail_transaksi
WHERE harga > 10000
ORDER BY harga
HAVING(pesanan >= 2)
GROUP BY id_detail, id_transaksi, id_menu, qty, harga;
```

6. Perbaiki SELECT statement di atas sehingga dapat dieksekusi dan menghasilkan *result-set* yang diinginkan.
7. Analisa SELECT statement di bawah ini. Paparkan alasan kenapa hasil eksekusi SELECT statement berikut ini mengembalikan pesan error!

```
SELECT id_detail, id_transaksi AS transaksi, id_menu AS menu, harga,
qty AS pesanan

FROM detail_transaksi

GROUP BY id_detail, menu, harga, transaksi

ORDER BY harga;
```

8. Perbaiki SELECT statement di atas sehingga dapat dieksekusi dan menghasilkan *result-set* yang diinginkan
9. Pada SELECT statement nomor 7, tambahkan aturan yaitu: hanya menampilkan baris yang sudah melakukan transaksi minimal dua kali
10. Kemudian tambahkan lagi aturan yaitu: hanya menampilkan baris yang harganya 12000 ke atas
11. Urutkan berdasarkan kolom harga secara ASCENDING

1.5 Tugas

1. Buat laporan praktikum dari percobaan yang dilakukan, masing-masing lengkapi dengan narasi, *query*, dan *screenshoot* hasil eksekusi. Tuliskan rangkuman dengan lengkap, runtut, dan mudah dipahami
2. Buat kesimpulan tentang apa yang anda dapatkan/pahami dari praktikum untuk materi **Fase Pemrosesan *Query* di SQL** yang baru dilakukan, sertakan kesimpulan tersebut dalam rangkuman anda

BAB II DATA RETRIEVAL

2.1 Tujuan

Setelah praktikum ini usai, mahasiswa diharapkan:

- mampu melakukan *data retrieval* menggunakan SELECT statement sesuai dengan kondisi-kondisi tertentu
- mampu mengimplementasikan penggunaan klausa DISTINCT pada SELECT statement
- mampu mengimplementasikan penggunaan operator aritmatika, operator perbandingan, operator logika, dan operator set pada SELECT statement; serta SELECT statement dengan kombinasi dari operator-operator tersebut

2.2 Dasar Teori

Data retrieval (pengambilan data) pada SQL dilakukan menggunakan SELECT statement. Ulasan ini akan membahas beberapa bentuk *data retrieval* yang bisa dilakukan pada basisdata anda menggunakan operator dan kondisi yang beragam.

Query berikut akan menampilkan seluruh kolom dan baris pada tabel `products`

```
SELECT * FROM products;
```

SELECT* pada contoh di atas berarti memilih semua kolom pada tabel. ALL atau * (*asterisk*) adalah eskpresi yang bisa digunakan untuk memanggil seluruh kolom pada tabel tanpa harus menuliskan nama kolom satu persatu. Beberapa resiko penggunaan asterisk antara lain:

- *unnecessary load network*, adanya load yang tidak penting terjadi pada jaringan
- *query is hard to index*, data yang cukup besar menjadikan *query* kesulitan untuk dijadikan index

```
SELECT TOP 3 * FROM products;
```

SELECT statement di atas akan menampilkan seluruh kolom pada tabel `customers`, adapun jumlah baris yang ditampilkan yakni 3 baris teratas sesuai dengan urutan primary key-nya.

Berikut ini adalah SELECT statement yang menghasilkan *result-set* serupa dengan SELECT statement di atas

```
SELECT * FROM products LIMIT 3;
```

Klausa TOP dan LIMIT pada SELECT statement digunakan untuk membatasi jumlah baris yang ingin ditampilkan. Klausa TOP digunakan di SQL Server dan MS Access, LIMIT digunakan di MySQL.

Menampilkan keseluruhan baris dari kolom tertentu dari sebuah tabel

```
SELECT product_id, name, weight, price FROM products;
```

Menampilkan baris data dengan nilai unik

```
SELECT DISTINCT(province) FROM customers;
```

DISTINCT adalah klausa yang digunakan untuk mendapatkan baris unik dari kolom. SELECT statement di atas akan mengembalikan baris unik pada kolom province, jika ada dua baris memiliki nilai yang sama, maka yang ditampilkan hanya satu saja. Misalnya, province memiliki 3 buah baris bernilai DIY, maka yang ditampilkan hanya satu buah baris bernilai DIY.

Tebel 2. 1 Kolom `province` dengan baris bernilai sama

province
DIY
DIY
DIY

Hasil SELECT DISTINCT-nya adalah sebagai berikut:

Tebel 2. 2 Hasil SELECT DISTINCT dari kolom di tabel 2.1

province
DIY

Hanya satu baris berbeda saja yang akan ditampilkan pada SELECT DISTINCT. Baris ini bersifat *non-case sensitive*, huruf besar atau kecil tidak dipermasalahkan. “DIY”, dan “diy” dianggap teks yang sama.

OPERATOR

Operator adalah kata atau karakter khusus yang dikenali oleh SQL untuk digunakan pada klausa WHERE dan/atau HAVING pada DDL atau DML *statement*. Ada beberapa macam operator pada SQL, yaitu: operator aritmatik (*aritmatic operator*), operator perbandingan (*comparison operator*), operator logika (*logical operator*), dan operator spesial (*special operator*).

2 Operator aritmatik (*aritmatic operator*)

Tebel 2. 3 Daftar operator aritmatik

Operator	Deskripsi
+	operator untuk penambahan
-	operator untuk pengurangan
*	operator untuk perkalian
/	operator untuk pembagian

Contoh:

```
SELECT name, price, stock, price*stock AS max_earning FROM products;
```

SELECT *statement* di atas akan mengembalikan kolom name, price, stock, dan max_earning, nilai baris di kolom max_earning berasal dari perkalian antara kolom price dan stock di tabel products.

Operator aritmatik ditujukan untuk melakukan perhitungan matematika sehingga hanya bisa dilakukan untuk kolom dengan tipe data numerik, seperti: INT, DOUBLE, FLOAT, dan lain-lain. Jika perhitungan matematika dipaksakan pada kolom non-numerik, maka *output* dari SELECT *statement* tersebut adalah 0.

3 Operator perbandingan (*comparison operator*)

Tebel 2. 4 Daftar operator perbandingan (*comparison operator*)

Operator	Deskripsi
=	sama dengan
>	lebih besar
<	lebih kecil
>=	lebih besar sama dengan
<=	lebih kecil sama dengan
<>	tidak sama dengan, biasanya ditulis pula dengan format → !=

Contoh:

```
SELECT * FROM customers WHERE country = "Sleman";
```

SELECT statement di atas akan mengembalikan seluruh kolom di tabel customers dan baris yang memenuhi kondisi: kolom country **sama dengan** Sleman.

```
SELECT * FROM reviews WHERE product_id <> "AV002002";
```

SELECT statement di atas akan mengembalikan seluruh kolom di tabel customers dan baris di mana product_id **selain** atau **tidak sama** dengan AV002002.

4 Operator logika (*logical operator*)

Berikut ini adalah daftar operator logika:

Tebel 2. 5 Daftar operator logika

Operator	Deskripsi
AND	Menyajikan *records* jika semua kondisi yang dipisahkan AND bernilai benar

OR	Menyajikan *record* jika salah satu kondisi yang dipisahkan OR bernilai benar
NOT	Menyajikan *record* jika kondisinya bernilai tidak benar (NOT TRUE)
LIKE	Menyajikan *record* sesuai dengan pattern yang diinginkan
BETWEEN	Menyajikan *record* berdasarkan *range* atau jangkauan tertentu
IN	Menyajikan *record* untuk *multiple value* bernilai TRUE

AND

Klausa AND digunakan untuk menyaring *record* jika kondisi pada klausa WHERE lebih dari satu, baris akan ditampilkan jika seluruh kondisi antara AND terpenuhi (TRUE).

Sintaks dasar – AND

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Contoh (1) Operator AND pada klausa WHERE

```
SELECT customer_id, full_name, province, DATE(join_date) AS join_date
FROM customers
WHERE province = "DIY" AND DATE(join_date) = "2022-06-27";
```

SELECT statement di atas akan mengembalikan kolom `customer_id`, `full_name`, `province`, dan `join_date` dari tabel `customers` yang memenuhi kondisi di mana provinsinya adalah DIY dan join_date-nya adalah 2022-06-27. Jika hanya satu kondisi yang memenuhi, misalnya provinsi adalah DIY tetapi join_date-nya tidak sesuai, ataupun sebaliknya, maka baris tersebut tidak akan ditampilkan.

OR

Klausa OR digunakan untuk menyaring *record* jika kondisi pada klausa WHERE lebih dari satu, jika salah satu kondisi yang dipisahkan oleh OR terpenuhi maka baris akan ditampilkan.

Sintaks dasar – OR

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

Contoh (2) Operator OR pada klausa WHERE

```
SELECT * FROM customers
WHERE province = "DIY" OR postal_code LIKE '555%';
```

SELECT statement di atas akan mengembalikan seluruh *records* di tabel customers yang provinsinya adalah DIY atau kode posnya diawali dengan angka 555. Jika sebuah *record* pada tabel customers memenuhi salah satu kondisi WHERE tersebut, maka *records* tersebut akan ditampilkan.

NOT

Klausa NOT digunakan untuk menyaring *record* dengan nilai NOT TRUE, artinya hanya baris yang memiliki nilai yang tidak sama dengan nilai pada NOT yang akan ditampilkan.

Sintaks dasar – NOT

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Contoh (3) Operator NOT pada klausa WHERE

```
SELECT customer_id, full_name, province
FROM customers
WHERE NOT province = "Jambi";
```

SELECT *statement* di atas akan mengembalikan *record* untuk kolom customer_id, full_name, dan province selain yang nilai provinsinya adalah Jambi.

Contoh (4) Kombinasi AND dan OR pada klausa WHERE

```
SELECT customer_id, full_name, province, DATE(join_date) AS join_date
FROM customers
WHERE province = "DIY" AND (Country = "Sleman" OR country = "Gn. Kidul");
```

SELECT statement di atas akan mengembalikan *records* yang memenuhi kondisi:

- provinsi DIY dan country Sleman, atau
- provinsi DIY dan country Gn. Kidul

Bagian pada kondisi WHERE yang menggunakan kurung buka dan tutup adalah bagian yang akan dieksekusi lebih dahulu (prioritas).

LIKE

Operator LIKE digunakan pada klausa WHERE untuk mendapatkan *pattern* atau pola tertentu saat pencarian. *Wildcard character* adalah karakter yang digunakan sebagai *subtitute* atau pengganti pada operator LIKE, *wildcard character* antara lain adalah:

- % (*percentage symbol*), merepresentasikan satu atau lebih karakter
  Contoh: ar% → arah, areh, area, arus, array
- _ (*underscore*), merepresentasikan satu buat karakter
  Contoh: a_r → air; a__s arus; h__i hari, hati

Berikut ini cara penggunaan LIKE dan *wildchard character* untuk mendapatkan *pattern* tertentu

Tebel 2. 6 Daftar cara penggunaan LIKE dan *wildcard character*

LIKE Operator	**Deskripsi**
LIKE "s%"	Mencari nilai dengan awalan s
LIKE '%s'	Mencari nilai dengan akhiran s
LIKE '%or%'	Mencari nilai yang mengandung or di posisi manapun or-nya
LIKE '_s%'	Mencari nilai dengan huruf s sebagai kata kedua

LIKE 'i_%'	Mencari kata yang diawali dengan huruf i dengan panjang minimal dua kata
LIKE 'i__%'	Mencari kata yang diawali dengan huruf i dengan panjang minimal tiga kata
LIKE 's%i'	Mencari kata dengan awalan s dan akhirnya i

Sintaks dasar – LIKE

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

Contoh (5) Klausa LIKE untuk mendapatkan *record* awalan tertentu

```
SELECT name, price FROM products
WHERE name LIKE "Mirac%";
```

Contoh (6) Klausa LIKE untuk mendapatkan *record* yang mengandung teks tertentu dikombinasikan dengan operator AND

```
SELECT name, price FROM products
WHERE name LIKE "%Skin%" AND name LIKE "Bae%";
```

BETWEEN

Operator BETWEEN akan menyaring *record* berdasarkan *range* dari nilai yang diberikan. Nilainya yang dibolehkan adalah numerik, date, dan texts.

Sintaks dasar – BETWEEN

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2
```

Contoh (7) Operator BETWEEN untuk tipe data DATE

```
SELECT name, province FROM customers
WHERE join_date BETWEEN '2022-02-01' AND '2022-12-01';
```

SELECT statement di atas akan menampilkan isi tabel customers dengan dua buah kolom yaitu name, dan province, dan baris data dengan nilai di kolom join_date antara 2022-02-01 sampai 2022-12-01.

Contoh (8) Operator BETWEEN untuk tipe data INT

```
SELECT name, stock, price FROM products
WHERE stock BETWEEN 10 AND 300;
```

SELECT statement di atas akan menampilkan data dari tabel `products` dengan tiga buah kolom yaitu `name`, `stock`, dan `price`. Baris data yang ditampilkan hanya baris yang memiliki stok antara 10 sampai dengan 300.

IN

Operator IN digunakan untuk men-spesifikasi *multiple value* pada klausa WHERE, operator ini adalah bentuk pendek dari klausa OR

Sintaks dasar – IN

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Contoh (9) Klausa IN dengan 3 buah nilai

```
SELECT name, stock, price FROM products

WHERE stock IN (10, 20, 30);
```

SELECT statement di atas akan menampilkan baris untuk kolom name, stock, dan price dari tabel products yang memiliki stock 10, atau 20, atau 30. Baris yang memenuhi salah satu nilai stock tersebut akan ditampilkan.

Klausa IN juga bisa digunakan untuk *subQuery*, sintaks dasarnya sebagai berikut:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT statement);
```

Berbeda dengan sintaks dasar pertama yang menerima nilai, pada sintaks di atas IN akan menerima sebuah SELECT statement.

Contoh (10) Klausa IN dengan *subquery*

```
SELECT name, stock, price FROM products
WHERE product_id IN (SELECT product_id FROM products WHERE stock >= 30);
```

SELECT statement di atas akan menyajikan data dari kolom `name`, `stock`, dan `price` dari tabel `products` dari seluruh baris yang stoknya bernilai lebih besar sama dengan 30.

5 Operator set (*set operator*)

Tebel 2. 7 Daftar operator set (*set operator*)

Operator	**Deskripsi**
UNION	Mengembalikan *result-set* dari dua buah SELECT statement atau lebih, baris yang sama hanya akan ditampilkan satu kali
UNION ALL	Seperti UNION tetapi menghiraukan nilai yang duplikat
EXCEPT	Mengembalikan baris dari SELECT statement pertama yang tidak ada kesamaannya dengan SELECT statement kedua
INTERSECT	Mengembalikan baris dari SELECT statement pertama hanya yang

	memiliki kesamaan nilai dengan baris SELECT statement kedua

UNION dan UNION ALL

Operator UNION digunakan untuk mengombinasikan *result-set* dari dua buah SELECT statement atau lebih, tanpa mengembalikan baris yang duplikat. Syarat menggunakan UNION antara lain:

- masing-masing SELECT statement memiliki jumlah kolom yang sama untuk ditampilkan
- kolom antar SELECT statement tersebut memiliki tipe data yang sama, dan barisnya memiliki urutan yang sama pula

Sintaks dasar – UNION dan UNION ALL

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
UNION
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
UNION ALL
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Contoh (1) UNION

```
SELECT province FROM customers
UNION
SELECT name FROM provinces;
```

SELECT statement di atas akan mengembalikan *result-set* dari kombinasi antara tabel `customers` kolom `province` dan tabel `provinces` kolom `name`, baris yang ditampilkan adalah semua baris unik yang ada pada masing-masing tabel

province
DIY
Jambi
Gorontalo
Kalimantan Selatan
DKI Jakarta
Jawa Tengah

Gambar 2. 1 *Result-set* dari contoh (1)

Contoh (2) UNION ALL

```
SELECT province FROM customers

UNION ALL

SELECT name FROM provinces;
```

Berbeda dengan SELECT statement di contoh (1), pada contoh (2) di atas SELECT statement akan mengembalikan *result-set* tanpa mengecek keunikannya, UNION ALL cenderung akan mengembalikan jumlah baris lebih banyak ketimbang UNION

province
DIY
DIY
DIY
Jambi
Gorontalo
Kalimantan Selatan
Kalimantan Selatan
DKI Jakarta
DKI Jakarta
DKI Jakarta
Jawa Tengah

Gambar 2. 2 *Result-set* dari contoh (2)

EXCEPT dan INTERSECT

Klausa EXCEPT digunakan untuk mengembalikan semua baris di SELECT statement pertama yang baris tersebut tidak ada pada SELECT statement kedua. Sementara itu, klausa INTERSECT digunakan untuk mengombinasikan dua buah SELECT statement, hanya baris di SELECT statement pertama yang memiliki kesamaan dengan SELECT statement kedua yang akan ditampilkan. Klausa INTERSECT adalah kebalikan dari EXCEPT.

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
EXCEPT
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
INTERSECT
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Contoh (3) EXCEPT

```
SELECT province FROM customers
EXCEPT
SELECT name FROM provinces;
```

SELECT statement di atas akan mengembalikan *result-set* dari SELECT statement pertama yang **tidak memiliki kesamaan** nilai dengan *result-set* di SELECT statement kedua.

province
Kalimantan Selatan
DKI Jakarta
Jawa Tengah

Gambar 2. 3 *Result-set* untuk contoh (3)

Contoh (4) INTERSECT

```
SELECT province FROM customers
INTERSECT
SELECT name FROM provinces;
```

province
DIY
Jambi
Gorontalo

Gambar 2. 4 *Result-set* untuk contoh (4)

2.3 Alat dan Bahan

Database tools	MySQL
File import	product_managements.sql

2.4 Langkah Percobaan

Berikut ini adalah percobaan-percobaan yang akan:

Praktikum 1 | Menampilkan seluruh data (kolom dan baris) menggunakan *asterisk*

Tampilkan seluruh data dari tabel `products`. Kemudian tampilkan 3 baris pertama *record* di tabel `products`

product_id	name	size	unit	weight	price	stock	discount_percentage	halal_mui	cruelty_free_status	age_usage_from	main_ingredient	how_to_use	concern_id
AV002002	Avoskin Miraculous Refining Toner	100	ml	200gr	195000	20	0	NA18181207654	YES	16	AHA-BHA-PHA + Nicinamide + 2% Tea Tree + Witch Haz...	Follow the steps of use and the following tips...	4
AV002102	Avoskin Retinol Toner	100	ml	350gr	179000	35	0	NA18211205744	YES	20	Water, Propylene Glycol, Niacinamide, Glycerin, Po...	Follow these steps and tips to get the benefits ...	3
AV002201	Avoskin YSB Alpha Arbutin 3% + Grapeseed	30	ml	150gr	139000	200	0	NA18202000279	YES	15	Aqua, Alpha-Arbutin, Butylene Glycol, Glycerin, Hy...	Apply a few drops to the face in the morning and e...	1
	Avoskin YSB Glow										Water, Glycerin, Propanediol	Pagi: Oleskan ke seluruh	

Praktikum 2 | Melakukan perhitungan antar kolom menggunakan operator aritmatik

Tampilkan omzet maksimum yang didapat dari penjualan dari setiap produk.

Omzet = harga dikali stok.

Tabel `products`, kolom yang ditampilkan: `name`, `price`, `stock`, dan `omzet_max`

Output:

name	price	stock	omzet_max
Avoskin Miraculous Refining Toner	195000	20	3900000
Avoskin Retinol Toner	179000	35	6265000
Avoskin YSB Alpha Arbutin 3% + Grapeseed	139000	200	27800000
Avoskin YSB Glow Concentrate Treatment 2	259000	160	41440000
Avoskin YSB Niacinamide 12% + Centella Asiatica	139000	70	9730000
Avoskin Natural Sublime Facial Cleanser	119000	2300	273700000

Praktikum 3 | Menggunakan DISTINCT untuk menampilkan baris yang unik

Tampilkan seluruh baris di kolom `province` di tabel `customers`

province
DIY
DIY
DIY
Jambi
Gorontalo
Kalimantan Selatan
Kalimantan Selatan
DKI Jakarta
DKI Jakarta
DKI Jakarta
Jawa Tengah
DIY

Tampilkan hanya baris unik saja dari kolom `province` di tabel `customers`

province
DIY
Jambi
Gorontalo
Kalimantan Selatan
DKI Jakarta
Jawa Tengah

Praktikum 4 | Menampilkan *record* dengan *pattern* tertentu

Tampilkan baris di tabel `product` yang nama produknya mengandung “YSB”

product_id	name
AV002201	Avoskin YSB Alpha Arbutin 3% + Grapeseed
AV002202	Avoskin YSB Glow Concentrate Treatment 2
AV002203	Avoskin YSB Niacinamide 12% + Centella Asiatica

Praktikum 5 | Menampilkan *record* dengan *range* tertentu

Tampilkan data produk (`product_id, name, stock`) yang memiliki stok antara 30 sampai 200

product_id	name	stock
AV002102	Avoskin Retinol Toner	35
AV002201	Avoskin YSB Alpha Arbutin 3% + Grapeseed	200
AV002202	Avoskin YSB Glow Concentrate Treatment 2	160
AV002203	Avoskin YSB Niacinamide 12% + Centella Asiatica	70

Praktikum 6 | Menggunakan kombinasi operator perbandingan, operator logika, dan klausa IN

Pilih tabel di basisdata product_managements.sql, buat dua buah SELECT statement yang mengandung kombinasi beberapa operator yang dibahas pada pertemuan ini.

2.5 Tugas

1. Buat laporan praktikum dari percobaan yang dilakukan, masing-masing lengkapi dengan narasi, *query*, dan *screenshoot* hasil eksekusi. Tuliskan rangkuman dengan lengkap, runtut, dan mudah dipahami
2. Buat kesimpulan tentang apa yang anda dapatkan/pahami dari praktikum untuk materi **Data Retrieval** yang baru dilakukan, sertakan kesimpulan tersebut dalam rangkuman anda.

BAB III SUBQUERY

3.1 Tujuan

Setelah praktikum ini usai, mahasiswa diharapkan:

- memahami konsep *subquery*
- mampu membedakan dan mengimplementasikan beberapa jenis *subquery*
- mampu membuat dan menampilkan data menggunakan single row sub query, multiple row subquey, multiple column subquery
- mampu memahami dan mengaplikasikan operator subquery yakni ANY, ALL, dan EXIST

3.2 Dasar Teori

Subquery bermakna sebuah *query* yang berada di dalam *query* lain. *Subquery* adalah SELECT statement yang disisipkan dalam sebuah SQL statement. Dengan menggunakan *subquery*, maka hasil dari *query* akan menjadi bagian dari *query* di atasnya. *Subquery* dapat ditempatkan dalam klausa FROM, WHERE, dan/atau HAVING. *Subquery* disebut juga ***sub-select***, ***nested select***, atau ***inner-select***, merupakan SELECT statement yang berada (*nested*) dalam terdapat perintah DML, bisa berupa: SELECT, INSERT, UPDATE atau DELETE.

Sintaks dasar:

```
SELECT column_name [, column_name ]
FROM table1 [, table2 ]
WHERE column_name OPERATOR
    (SELECT column_name [, column_name ]
    FROM table1 [, table2 ]
    [WHERE])
```

Berdasarkan *output* dari *subquery*, ada tiga jenis *subquery* yaitu:

- *single-row subquery*

subquery yang *inner-select*-nya hanya mengembalikan satu *record* (baris)

- *multiple-row subquery*

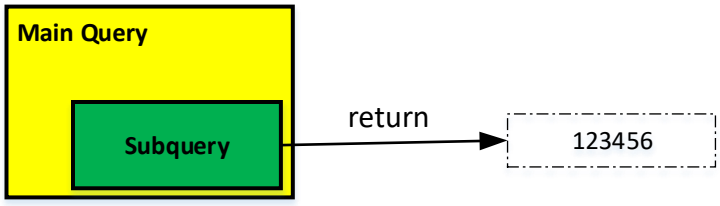
  subquery yang *inner-select*-nya mengembalikan lebih dari satu *record* (baris)

- *multiple-column subquery*

  query yang *inner-select*-nya mengembalikan lebih dari satu kolom

SINGLE-ROW SUBQUERY

Single-row subquery menghasilkan satu baris data dari perintah SELECT.



Gambar 3. 1 Ilustrasi *single-row subquery*

Ilustrasi di atas menggambarkan bahwa *return value* dari *subquery* adalah satu buah baris data. Adapun operator yang dapat digunakan dalam penerapan *single-row subquery* dapat dilihat pada tabel berikut ini.

Tabel 3. 1 Operator single-row subquery

Operator	**Deskripsi**
=	sama dengan
>	lebih besar
<	lebih kecil
>=	lebih besar sama dengan
<=	lebih kecil sama dengan
<>	tidak sama dengan, biasanya ditulis pula dengan format → !=

Contoh berikut ini disajikan tabel variasi dengan beberapa *field* didalamnya.

Gambar 3. 2 Tabel Variasi

Contoh (1) SELECT statement dengan *single row subquery*

```
SELECT nama AS menu, harga_tambahan
FROM variasi
WHERE harga_tambahan >
        (SELECT harga_tambahan FROM variasi WHERE id_variasi = 'V003');
```

SELECT statement di atas akan menampilkan data dari kolom `nama` dan `harga_tambahan` yang memenuhi kondisi yaitu harganya lebih besar dari variasi dengan `id_variasi` = 'V003'. Gambar berikut ini adalah *output* dari SELECT statement di atas:

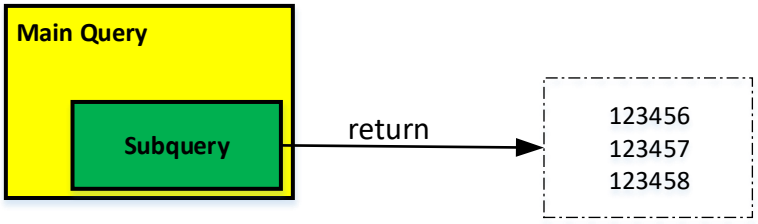
	Menu	harga_tambahan
1	EXTRA BAKSO	2000
2	EXTRA AYAM	3000
3	EXTRA KUPAT TAHU	1500
4	EXTRA NASI	1500
5	EXTRA SOMAY	2500
6	EXTRA SATE	3500

Gambar 3. 3 Hasil eksekusi contoh (1)

MULTIPLE-ROW SUBQUERY

Multiple-row subquery adalah *subquery* yang mengembalikan lebih dari satu baris data. *Multiple-row subquery* dapat menggunakan operator komparasi IN, ANY/SOME, atau ALL.

Ilustrasi *multiple-row subquery* dapat ditunjukan pada gambar berikut ini:



Gambar 3. 4 Ilustrasi *Multiple -row subquery*

Ilustrasi di atas menggambarkan *return value* dari *multiple row subquery* dapat berupa dua atau lebih baris data, dengan hanya satu kolom.

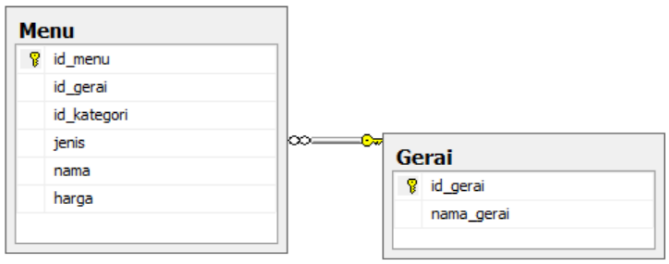
Tabel 3. 2 Tabel nilai_akhir

KODE	NILAI
A	100
B	150
C	200
D	250
E	300
F	350
G	370

Contoh (2) *Multiple-row subquery* dengan klausa IN

```
SELECT nilai
FROM nilai_akhir
WHERE nilai IN (
   SELECT nilai FROM nilai_akhir WHERE kode IN ('B', 'D', 'E'));
```

SELECT statement berikut ini akan menampilkan *record* data pada tabel xzy dengan kode B,D,E.



Gambar 3. 5 Struktur tabel menu dan gerai

Contoh (3) *Multiple-row subquery* dengan operator ANY

```
SELECT id_menu, harga, id_gerai
FROM menu
WHERE harga = ANY(SELECT MAX(harga) FROM menu);
```

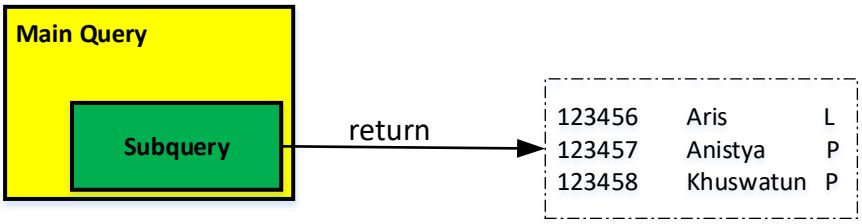
SELECT statement berikut ini akan menampilkan *record* menu dengan nilai harga maksimum dari tabel menu dan pemilik gerainya. *Output SELECT statement* di atas adalah sebagai berikut:

	id_menu	harga	id_gerai
1	MK07	18000	G007

Gambar 3. 6 Hasil eksekusi contoh (2)

MULTIPLE-COLUMN SUBQUERY

Multiple-column subquery menghasilkan lebih dari satu baris dan satu kolom data.



Gambar 3. 7 Ilustrasi *Multiple-column subquery*

Ilustrasi *multiple-column subquery* dapat dilihat pada gambar di atas. Sebagaimana namanya, *multiple-column subquery* mengembalikan nilai dalam bentuk *result-set*, yakni sebuah tabel dengan beberapa baris data dan beberapa kolom.

Contoh (4) *Multiple-column subquery* dengan *subquery* di bagian SELECT

```
SELECT nama AS menu, (SELECT SUM(harga) FROM detail_transaksi
WHERE id_menu = menu.id_menu)
AS [AMOUNT] FROM menu
ORDER BY AMOUNT DESC;
```

Adapun hasil dari perintah SELECT statement tersebut menampilkan *record* kolom `menu` dan `amount`:

	MENU	AMOUNT
1	SOMAY	116000
2	NASI PADANG	72000
3	MIE AYAM	60000
4	BAKSO	44000
5	AYAM MADU 2	35000
6	SATE	34000
7	KUPAT TAHU	27000
8	KOPI	25000
9	WEDANG UWOH	15000
10	JUS BUAH	12000

Gambar 3. 8 Hasil eksekusi *multiple-cloumn subquery*

Hasil eksekusi dari contoh (4) di atas adalah sebuah tabel dengan dua buah kolom yakni `menu` dan `amount`, kolom amount berisi jumlah harga dari masing-masing menu, tabel tersebut berisi 10 baris data.

Subquery juga dapat dikombinasikan dengan fungsi agregasi, perhatikan contoh berikut ini:

Contoh (5) fungsi agregasi dalam *subquery*

```
SELECT name, salary
FROM staff
WHERE salary > (SELECT AVG(salary) FROM staff));
```

SELECT statement di atas akan menampilkan kolom `name` dan `salary` dari tabel `staff` dengan ketentuan hanya staff yang memiliki salary di atas rata-rata saja yang barisnya ditampilkan.

OPERATOR SUBQUERY

Ada beberapa operator khusus yang digunakan dalam sebuah SQL statement dengan *subquery*. Operator tersebut adalah ANY, ALL, dan EXISTS.

Tabel 3. 3 Operator *subquery*

Operator	**Keterangan**
ANY	Mengembalikan nilai boolean, TRUE jika salah satu subquery memenuhi kondisi, jika tidak ada satu barispun yang memenuhi kondisi maka FALSE
ALL	Mengembalikan nilai boolean, TRUE jika semua set subquery memenuhi kondisi, selain itu maka FALSE
EXISTS	Mengembalikan nilai boolean, TRUE jika subquery mengembalikan minimal satu baris, sebaliknya FALSE

ANY dan ALL

Sintaks dasar:

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

EXISTS

Sintaks dasar:

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS(SELECT column_name FROM table_name WHERE condition);
```

Contoh (6) *Subquery* dengan klausa EXISTS

```
SELECT nama, jenis, harga FROM menu
WHERE EXISTS (SELECT harga FROM detail_transaksi
WHERE id_menu = menu.id_menu AND harga < 10000);
```

SELECT statement di atas akan mengembalikan *record* yang terdiri dari kolom `nama`, `jenis`, dan `harga` dari baris yang memenuhi kondisi: memiliki `id_menu` yang sama di tabel `Menu` dan `Detail_Transaksi`, dan harganya kurang dari 10000.

	nama	jenis	harga
1	GORENGAN	Makanan	5000
2	JERUK	Minuman	2000
3	TEH	Minuman	2000
4	LEMONTIE	Minuman	2500
5	LEMONTIE	Minuman	2500

Gambar 3. 9 Hasil *eksekusi contoh (6)*

3.3 Alat dan Bahan

Database tools	MySQL
File import	FoodsCenter.sql/.mdf

3.4 Langkah Percobaan

Berikut ini adalah percobaan-percobaan yang akan anda kerjakan:

Praktikum 1 | Menampilkan data dengan single-row

1. Buatlah perintah untuk menampilkan metode pembayaran serta tanggal dari karyawan ‘Sandra’.

	metode_pembayaran	tanggal
1	CAST	2022-01-04 00:00:00.000
2	CAST	2022-01-05 00:00:00.000
3	GO-PAY	2022-01-06 00:00:00.000
4	GO-PAY	2022-02-03 00:00:00.000
5	CAST	2022-02-03 00:00:00.000
6	GO-PAY	2022-04-06 00:00:00.000
7	CAST	2022-04-03 00:00:00.000
8	CAST	2022-05-03 00:00:00.000
9	CAST	2022-05-04 00:00:00.000

2. Buatlah perintah untuk menampilkan informasi transaksi id_transaksi dan tanggal transaksi pelanggan dengan id_meja ’M05P’.

	id_transaksi	id_meja	tanggal
1	TR03	M05P	2022-01-04 00:00:00.000
2	TR04	M05P	2022-01-05 00:00:00.000
3	TR05	M05P	2022-01-05 00:00:00.000
4	TR15	M05P	2022-02-28 00:00:00.000
5	TR21	M05P	2022-04-05 00:00:00.000
6	TR31	M05P	2022-02-28 00:00:00.000
7	TR37	M05P	2022-01-04 00:00:00.000
8	TR38	M05P	2022-01-05 00:00:00.000
9	TR39	M05P	2022-04-05 00:00:00.000

3. Buatlah perintah untuk menampilkan data menu yang jumlah harganya diatas menu ‘nasi padang’.

	nama	harga
1	SOMAY	15000
2	KUPAT TAHU	13500
3	NASI GORENG	13000
4	AYAM MADU 2	18000
5	SATE	17000
6	KOPI	12500
7	WEDANG UWOH	15000

4. Buatlah perintah untuk menampilkan data nama, jenis dan harga dari menu yang berjenis minuman.

	nama	harga
1	KOPI	12500
2	JERUK	2000
3	TEH	2000
4	WEDANG UWOH	15000
5	JERUK	2000
6	JUS BUAH	12000
7	LEMONTIE	2500
8	MINUMAN KALENG	7500
9	LEMONTIE	2500

5. Buatlah perintah untuk menampilkan data transaksi pada id transaksi ‘TR37’.

	id_transaksi	id_karyawan	id_meja	id_pelangan	id_variasi	tanggal	metode_pembayaran	status_transaksi
1	TR37	KY10	M05P	PG03	V003	2022-01-04 00:00:00.000	GO-PAY	SUKSES

Praktikum 2 | Menampilkan data dengan multiple-row

1. Buatlah perintah untuk menampilkan data menu yang sudah pernah terjual oleh pelanggan.

	id_menu	nama
1	MK01	BAKSO
2	MK02	SOMAY
3	MK03	MIE AYAM
4	MK04	KUPAT TAHU
5	MK06	NASI PADANG
6	MK07	AYAM MADU 2
7	MK09	SATE
8	MK10	GORENGAN
9	MK11	KOPI
10	MK12	JERUK
11	MK13	TEH
12	MK14	WEDANG U...
13	MK16	JUS BUAH
14	MK19	LEMONTIE
15	MK21	LEMONTIE

2. Buatlah perintah untuk menampilkan data menu yang ada di Gerai ‘Somay Mentes’.

	id_menu	id_gerai	id_kategori	jenis	nama	harga
1	MK02	G005	KT05	Makanan	SOMAY	15000

3. Buatlah perintah untuk menampilkan data pelanggan yang sudah pernah membeli.

	id_pelangan	nama	no_hp
1	PG01	NABILA	89646599868
2	PG02	ANISA	82235899147
3	PG03	NEBULA	85156086291
4	PG04	SINDHI	81326582921
5	PG05	DIMAS	88239763254
6	PG06	RAFLY	87837953418
7	PG07	TEDDY	85239586625
8	PG08	MARIO	82235291227
9	PG09	GHALY	85929206670
10	PG10	KHARI...	82314012027
11	PG12	MOMO	85861187626
12	PG13	ALBERT	85641122582
13	PG14	ADIT P...	87772407293

4. Buatlah perintah untuk menampilkan data nama variasi yang tidak terdapat pada tabel transaksi.

	id_variasi
1	V002
2	V004
3	V008

5. Buatlah perintah untuk menampilkan data meja pesan yang belum pernah melakukan digunakan.

	id_meja	no_meja	fasilitas	nama_ruang
1	M03P	MP06	25 KURSI LUAR	RAMA

Praktikum 3 | Menampilkan data dengan fungsi agregasi

1. Buatlah perintah untuk menampilkan data pelanggan yang melakukan lebih dari 4 kali transaksi.

	nama	jumlah transaksi
1	SINDHI	6
2	TEDDY	8

2. Buatlah perintah untuk menampilkan data karyawan dan berapa kali sudah melayani pelanggan.

	nama	jumlah layanan
1	AYUNDA	8
2	BAGAS	6
3	INDRA	4
4	MAHARANI	10
5	SANDRA	12

3. Buatlah perintah untuk menampilkan total jumlah transaksi masing masing pelanggan.

	customer	total
1	ADIT PERMADI	12000
2	ALBERT	42000
3	ANISA	63000
4	DIMAS	36000
5	GHALY	15000
6	KHARISMA	24000
7	MARIO	72500
8	MOMO	65500
9	NABILA	51500
10	NEBULA	90000

4. Buatlah perintah untuk menampilkan data pelanggan yang paling sering melakukan transaksi.

	id_pelangan	nama	no_hp
1	PG07	TEDDY	85239586625

5. Buatlah perintah untuk menampilkan data nama pelanggan,nomor meja, dan status pembayaran serta berapa banyak pelanggan tersebut bertransaksi pada tangal '2022-01-04'.

	berapa kali	nama	id_meja	metode_pembayaran	status_transaksi
1	1	ANISA	M04P	CAST	SUKSES
2	1	NABILA	M04P	CAST	SUKSES
3	2	NEBULA	M05P	GO-PAY	SUKSES

6. Buatlah perintah untuk menampilkan data gerai berapa banyak yang sudah terjual.

	Nama_Gerai	Jumlah_Terjual
1	AYAM CEMANY MREBES	2
2	BAKSO BAROKAH	2
3	GORENG SRENG	1
4	KEDAI SEGAR	4
5	KUPAK TAHU BLALAK	2
6	MIE AYAM LARIS	1
7	NASGOS RAMINTEN	2
8	PADANG JAYA	2
9	SATE KELINCI	2
10	SEGO TAMBAH	2

7. Buatlah perintah untuk menampilkan data nama pelanggan, metode pembayaran dan status pembayaran serta berapa banyak pelanggan tersebut bertransaksi dan berapa total pembelian mereka diatas 30000 dan diurutkan dari yang terkecil.

	nama	metode_pembayaran	status_transaksi	belanja	Total
1	SINDHI	CAST	SUKSES	6	89500
2	TEDDY	CAST	SUKSES	5	80000
3	MOMO	CAST	SUKSES	3	65500
4	ANISA	CAST	SUKSES	3	63000
5	NEBULA	GO-PAY	SUKSES	2	60000
6	NABILA	CAST	SUKSES	3	51500
7	TEDDY	GO-PAY	SUKSES	3	46000
8	ALBERT	GO-PAY	SUKSES	3	42000
9	MARIO	CAST	SUKSES	2	40500
10	DIMAS	CAST	SUKSES	1	36000

Praktikum 4 | Menampilkan data dengan Operator EXIST

1. Buatlah perintah untuk menampilkan data jenis makanan dan minuman yang memiliki harga < 12000

	nama	jenis	harga
1	BAKSO	Makanan	11000
2	SOMAY	Makanan	15000
3	GORENGAN	Makanan	5000
4	JERUK	Minuman	2000
5	TEH	Minuman	2000
6	LEMONTIE	Minuman	2500
7	LEMONTIE	Minuman	2500

2. Buatlah perintah untuk menampilkan data id makanan dan minuman yang pernah membeli >1.

	Menu	Jumlah_Pembelian
1	MK09	2
2	MK02	2
3	MK06	3
4	MK01	3
5	MK03	2
6	MK09	2
7	MK02	2
8	MK06	3
9	MK01	2
10	MK01	3
11	MK03	2
12	MK04	2

3. Buatlah perintah untuk menampilkan data kolom jenis, nama dan harga pada tabel menu pada gerai ‘Sate Kelinci’.

	jenis	nama	harga
1	Makanan	SATE	17000
2	Minuman	LEMONTIE	2500

Praktikum 5 | Menampilkan data dengan multiple-row atau multiple-column ??

1. Buatlah perintah untuk menampilkan no meja ‘MP02’,’MP06’,’MP08’.

	no_meja
1	MP02
2	MP06
3	MP08

2. Buatlah perintah untuk menampilkan data yang id_transaksi dan tanggal transaksi yang berada di ruang ‘SINTA’.

	id_transaksi	tanggal
1	TR10	2022-02-03 00:00:00.000
2	TR14	2022-02-22 00:00:00.000
3	TR16	2022-03-01 00:00:00.000
4	TR20	2022-03-05 00:00:00.000
5	TR29	2022-05-01 00:00:00.000

3. Buatlah perintah untuk menampilkan daftar harga minuman pada tabel menu.

	nama	harga
1	KOPI	12500
2	JERUK	2000
3	TEH	2000
4	WEDANG UWOH	15000
5	JERUK	2000
6	JUS BUAH	12000
7	LEMONTIE	2500
8	MINUMAN KALENG	7500
9	LEMONTIE	2500

4. Buatlah perintah untuk menampilkan data nama pelanggan yang melakuan transaksi pada tanggal 2022-05-04.

	id_pelangan	nama	no_hp
1	PG01	NABILA	89646599868
2	PG02	ANISA	82235899147

5. Buatlah perintah untuk menampilkan data makanan yang merupakan kategori olahan ayam.

	jenis	menu	harga
1	Makanan	AYAM MADU 2	18000

3.5 Tugas

1. Buat laporan praktikum dari percobaan yang dilakukan, masing-masing lengkapi dengan narasi, *query*, dan *screenshoot* hasil eksekusi. Tuliskan rangkuman dengan lengkap, runtut, dan mudah dipahami
2. Buat kesimpulan tentang a pa yang anda dapatkan/pahami dari praktikum untuk materi **Subquery** yang baru dilakukan, sertakan kesimpulan tersebut dalam rangkuman anda

BAB IV JOIN

4.1 Tujuan

Setelah praktikum ini usai, mahasiswa diharapkan:

- Mampu memahami konsep join
- mampu mengaplikasikan penggunaan *join* untuk menampilkan data dari beberapa tabel
- mampu membedakan beberapa jenis join berdasarkan cara penulisan, perbedaan hasil *output*, dan kegunaannya baik *inner join, cross join*, maupun *outer join*
- mampu mengaplikasikan *outer join* baik *left join*, *right join* dan/atau *full join* untuk menampilkan data dari beberapa tabel

4.2 Dasar Teori

JOIN

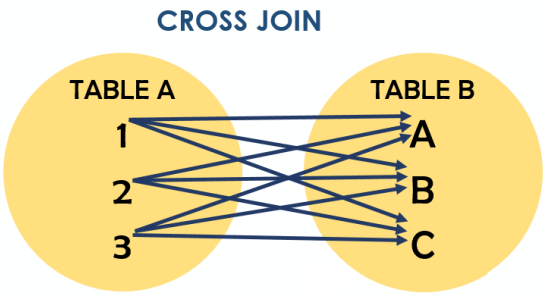
Join merupakan sebuah operasi yang digunakan untuk mendapatkan data gabungan dari dua tabel atau lebih. Operasi ini digunakan dalam perintah *SELECT* dan biasanya dipakai untuk memperoleh data secara detail dari tabel-tabel yang saling terkait (memiliki relasi).

Terdapat 3 jenis join yaitu :

1. ***Cross join (cartesian join)***
2. ***Inner join***
3. ***Outer join*****,** terdiri dari: *left join, right join,* dan *full outer join*

CROSS JOIN

Cross join atau *cartesian join* adalah *join* menghasilkan *output* berupa kombinasi dari semua baris yang terdapat dalam tabel-tabel yang digabungkan baik yang berpasangan maupun yang tidak berpasangan. Pada praktiknya *join* jenis ini jarang bahkan tidak pernah dipakai. Meskipun begitu, jenis *join* inilah yang merupakan dasar dari *join* antar tabel.

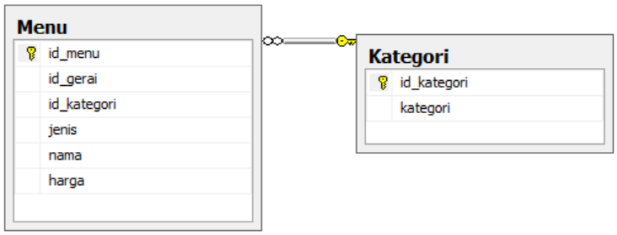


Gambar 4. 1 Ilustrasi cross join

Pada *cross join*, jumlah *record* yang dihasilkan sama dengan jumlah *record* tabel pertama dikali jumlah *record* tabel kedua.

Jumlah baris = jumlah baris di tabel pertama * jumlah baris di tabel kedua.

Jika jumlah baris tabel pertama = 1000, dan jumlah baris tabel kedua = 1000, maka hasil dari *cross join* tabel pertama dan kedua adalah 1000 x 1000 = 1.000.000 baris



Gambar 4. 2 Tabel menu dan kategori

Contoh (1) *Join* dua tabel menggunakan klausa *cross join* secara eksplisit

```
SELECT * FROM menu CROSS JOIN kategori;
```

Contoh (2) *Join* dua tabel, *cross join* tanpa klausa *cross join*

```
SELECT * FROM menu, kategori;
```

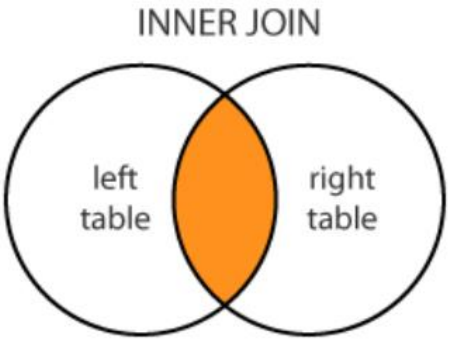
Adapun untuk hasil perintah tersebut

	id_menu	id_gerai	id_kategori	jenis	nama	harga	id_kategori	kategori
1	MK01	G001	KT06	Makanan	BAKSO	11000	KT01	OLAHAN AYAM
2	MK02	G005	KT05	Makanan	SOMAY	15000	KT01	OLAHAN AYAM
3	MK03	G003	KT03	Makanan	MIE AYAM	12000	KT01	OLAHAN AYAM
4	MK04	G008	KT07	Makanan	KUPAT TAHU	13500	KT01	OLAHAN AYAM
5	MK05	G006	KT02	Makanan	NASI GORENG	13000	KT01	OLAHAN AYAM
6	MK06	G002	KT02	Makanan	NASI PADANG	12000	KT01	OLAHAN AYAM
7	MK07	G007	KT01	Makanan	AYAM MADU 2	18000	KT01	OLAHAN AYAM
8	MK08	G004	KT08	Makanan	NASI SHOP	8000	KT01	OLAHAN AYAM
9	MK09	G009	KT04	Makanan	SATE	17000	KT01	OLAHAN AYAM
10	MK10	G010	KT09	Makanan	GORENGAN	5000	KT01	OLAHAN AYAM

Gambar 4. 3 *Result-set* contoh (1) dan (2)

INNER JOIN

Inner join adalah *join* yang menghasilkan *output* berupa kombinasi baris-baris yang memiliki pasangan saja. Kombinasi baris yang tidak berpasangan akan dieliminasi atau tidak ditampilkan pada *result-set*. Baris-baris yang tidak memiliki pasangan pada tabel lainnya juga tidak dimunculkan.



Gambar 4. 4 Ilustrasi inner join

Pada *inner join* hanya bagian yang beririsan saja (blok warna *orange*) yang akan ditampilkan. Bagian putih di *left table* dan *right table* adalah baris yang tidak memiliki pasangan, bagian ini tidak akan ditampilkan pada *result-set*.

Contoh (1) Format *inner join* dengan klausa INNER JOIN yang ditulis eksplisit

```
SELECT kategori.*, menu.*
FROM menu INNER JOIN kategori
ON kategori.id_kategori = menu.id_kategori;
```

Contoh (2) Padanan *inner join* dari contoh (1)

```
SELECT * FROM kategori, menu
WHERE kategori.id_kategori = menu.id_kategori;
```

SELECT statement di atas baik contoh 1 dan contoh 2 akan mengembalikan seluruh *record* kolom di tabel kategori dan menu. Adapun untuk hasil perintah tersebut :

	id_kategori	kategori	id_menu	id_gerai	id_kategori	jenis	nama	harga
1	KT06	OLAHAN BAKSO	MK01	G001	KT06	Makanan	BAKSO	11000
2	KT05	OLAHAN SOMAY	MK02	G005	KT05	Makanan	SOMAY	15000
3	KT03	OLAHAN MIE	MK03	G003	KT03	Makanan	MIE AYAM	12000
4	KT07	OLAHAN TAHU	MK04	G008	KT07	Makanan	KUPAT TAHU	13500
5	KT02	OLAHAN NASI	MK05	G006	KT02	Makanan	NASI GORENG	13000
6	KT02	OLAHAN NASI	MK06	G002	KT02	Makanan	NASI PADANG	12000
7	KT01	OLAHAN AYAM	MK07	G007	KT01	Makanan	AYAM MADU 2	18000
8	KT08	OLAHAN SHOP	MK08	G004	KT08	Makanan	NASI SHOP	8000
9	KT04	OLAHAN SATE	MK09	G009	KT04	Makanan	SATE	17000

Gambar 4. 5 Hasil eksekusi perintah inner join contoh (1) dan (2)

Catatan :

- Klausa INNER pada INNER JOIN dapat dihapuskan. Jadi, cukup menuliskan 'JOIN' saja maknanya sama dengan INNER JOIN
- Jika kolom yang ingin ditampilkan ada di dua tabel atau lebih, maka harus ditentukan tabel mana yang diinginkan untuk menghindari terjadinya error "*ambiguous column name*"
- Gunakan alias jika nama tabel terlalu panjang, sehingga mudah diingat, dan penulisan *query* menjadi lebih ringkas.

Contoh (3) *SELECT* statement *inner join* tanpa alias

```
SELECT menu.id_menu, nama, harga, kategori.kategori
FROM menu INNER JOIN kategori
ON menu.id_kategori=kategori.id_kategori;
```

Contoh (4) *SELECT* statement *inner join* dengan alias

```
SELECT m.id_menu, nama, harga, k.kategori
FROM menu m INNER JOIN kategori k
ON m.id_kategori= k.id_kategori;
```

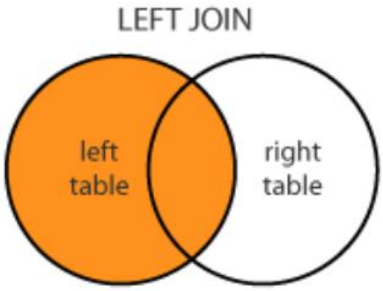
SELECT statement diatas sama-sama akan menghasilkan *record* yang sama, hanya saja dalam penulisan lebih ringkas dengan penggunaan alias. Di sini kita membuat beberapa nama alias seperti tabel menu menjadi m dan kategori menjadi k, penggunaan alias akan mempermudah penulisan SELECT statement dan mempercantik hasil tampilan SELECT statement.

OUTER JOIN

Outer join hampir sama dengan *inner join*, hanya saja baris yang tak memiliki pasangan tetap akan ditampilkan. Berdasarkan *output*-nya, ada 3 jenis *outer join* yaitu *left join*, *right join*, dan *full outer join*.

1. LEFT JOIN

LEFT JOIN akan menampilkan semua baris dari tabel pertama (*left table*) baik yang beririsan maupun yang tidak beririsan dengan tabel kedua (*right table*).



Gambar 4. 6 Ilustrasi left join

Pada *left join*, baris yang ditampilkan adalah bagian berwarna *orange* atau seluruh baris di tabel pertama (*left table*), termasuk baris yang memiliki pasangan dengan tabel kedua (*right table*) dan yang tidak memiliki pasangan.

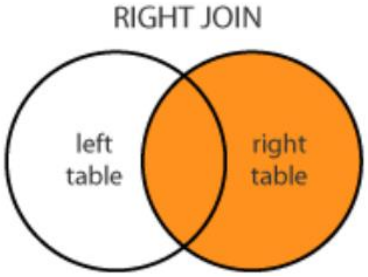
Contoh (5) SELECT statement dengan *left join*

```
SELECT gerai.nama_gerai, menu.nama, menu.jenis
FROM gerai LEFT JOIN menu
ON gerai.id_gerai=menu.id_gerai;
```

SELECT statement pada *left join* ini akan membuat parameter pada sebelah kiri, menampilkan *record* dari tabel `gerai` dan `menu`, jika ada *record* yang tidak memiliki pasangan maka akan berisi NULL.

2. RIGHT JOIN

RIGHT JOIN adalah kebalikan dari LEFT JOIN yaitu tabel yang menjadi acuan ada di sebelah kanan atau tabel kedua (*right table*).



Gambar 4. 7 Ilustrasi right join

Pada *right join*, baris yang ditampilkan adalah bagian berwarna *orange* atau seluruh baris di tabel kedua (*right table*), termasuk baris yang memiliki pasangan dengan tabel pertama (*left table*) dan yang tidak memiliki pasangan.

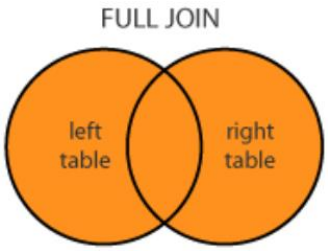
Contoh (6) SELECT statement dengan *right join*

```
SELECT pelanggan.nama, transaksi.tanggal, transaksi.metode_pembayaran
FROM pelanggan RIGHT JOIN transaksi
ON pelanggan.id_pelangan=transaksi.id_transaksi;
```

SELECT statement pada *right join* ini akan membuat parameter pada sebelah kanan, menampilkan *record* dari tabel `pelanggan` dan `transaksi`, pada statement ini akan menampilkan seluruh *record* dari tabel `pelanggan` baik yang memiliki pasangan dengan tabel `transaksi` maupun yang tidak memiliki pasangan. Baris yang tidak memiliki pasangan, maka bagian kosongnya akan diisi dengan NULL.

3. FULL JOIN

Full join menampilkan seluruh *record* dari dua tabel, baik *record* tersebut memiliki pasangan ataupun tidak.



Gambar 4. 8 Ilustrasi full join

Pada *full join*, baris yang ditampilkan adalah seluruh baris di tabel pertama (*left table*) dan tabel kedua (*right table*), termasuk baris yang tidak memiliki pasangan, baris yang tidak memiliki pasangan secara otomatis akan bernilai NULL.

Contoh (7) *SELECT statement* dengan *full join.*

```
SELECT p.id_pelangan, p.no_hp, t.tanggal, t.metode_pembayaran,
t.status_transaksi
FROM pelanggan p FULL JOIN transaksi t
ON p.id_pelangan = t.id_pelangan;
```

SELECT statement pada *full join* ini akan menampilkan *record* dari tabel `pelanggan` dan `transaksi`, baik *record* yang memiliki pasangan maupun *record* yang tidak memiliki pasangan.

4.3 Alat dan Bahan

MySQL atau SQL Server., import file FoodsCenter.sql/.mdf berikut di *database tools* anda

Database tools	MySQL
File import	FoodsCenter.sql/.mdf

4.4 Langkah Percobaan

Berikut ini adalah percobaan-percobaan yang akan dilakukan:

Praktikum 1 | Menampilkan data dengan INNER JOIN

1. Buatlah perintah untuk menampilkan id transaksi dan nama karyawan yang bertransaksi

	id_transaksi	nama
1	TR01	SANDRA
2	TR02	AYUNDA
3	TR03	BAGAS
4	TR04	MAHARANI
5	TR05	SANDRA
6	TR06	INDRA
7	TR07	SANDRA
8	TR08	BAGAS
9	TR09	MAHARANI
10	TR10	SANDRA

2. Buatlah perintah untuk menampilkan data gerai, menu dan harganya.

	nama_gerai	nama	harga
1	BAKSO BAROKAH	BAKSO	11000
2	SOMAY MENTES	SOMAY	15000
3	MIE AYAM LARIS	MIE AYAM	12000
4	KUPAK TAHU BLALAK	KUPAT TAHU	13500
5	NASGOS RAMINTEN	NASI GORENG	13000
6	PADANG JAYA	NASI PADANG	12000
7	AYAM CEMANY MREBES	AYAM MADU 2	18000
8	SEGO TAMBAH	NASI SHOP	8000
9	SATE KELINCI	SATE	17000
10	GORENG SRENG	GORENGAN	5000

3. Buatlah perintah untuk menampilkan data menu makanan dan jenis kategori.

	id_kategori	nama
1	KT06	BAKSO
2	KT05	SOMAY
3	KT03	MIE AYAM
4	KT07	KUPAT TAHU
5	KT02	NASI GORENG
6	KT02	NASI PADANG
7	KT01	AYAM MADU 2
8	KT08	NASI SHOP
9	KT04	SATE
10	KT09	GORENGAN

4. Buatlah perintah untuk menampilkan data menu yang jumlah transaksinya lebih dari 2

	nama
1	NASI PADANG
2	BAKSO
3	NASI PADANG
4	BAKSO
5	LEMONTIE

5. Buatlah perintah untuk menampilkan data pelanggan dan tanggal transaksinya.

	nama	tanggal
1	NABILA	2022-01-04 00:00:00.000
2	ANISA	2022-01-04 00:00:00.000
3	NEBULA	2022-01-04 00:00:00.000
4	SINDHI	2022-01-05 00:00:00.000
5	DIMAS	2022-01-05 00:00:00.000
6	RAFLY	2022-01-06 00:00:00.000
7	TEDDY	2022-01-06 00:00:00.000
8	MARIO	2022-02-02 00:00:00.000
9	GHALY	2022-02-02 00:00:00.000
10	KHARISMA	2022-02-03 00:00:00.000

6. Buatlah perintah untuk menampilkan data transaksi lengkap beserta nama karyawan, nomer meja, nama ruang, nama pelanggan, nama variasi dan harga tambahan variasi.

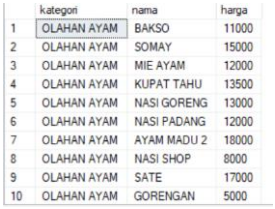
	id_transaksi	nama_karyawan	no_meja	nama_ruang	pelanggan	tambahan	harga_tambahan
1	TR01	SANDRA	MP08	BIMA	NABILA	NO EXTRA	0
2	TR02	AYUNDA	MP08	BIMA	ANISA	NO EXTRA	0
3	TR03	BAGAS	MP10	NAKULA	NEBULA	EXTRA KRUPUK	1000
4	TR04	MAHARANI	MP10	NAKULA	SINDHI	NO EXTRA	0
5	TR05	SANDRA	MP10	NAKULA	DIMAS	NO EXTRA	0
6	TR06	INDRA	MP02	SADEWA	RAFLY	NO EXTRA	0
7	TR07	SANDRA	MP02	SADEWA	TEDDY	NO EXTRA	0
8	TR08	BAGAS	MP08	BIMA	MARIO	EXTRA BAKSO	2000
9	TR09	MAHARANI	MP12	ARJUNA	GHALY	EXTRA SOMAY	2500
10	TR10	SANDRA	MP04	SINTA	KHARISMA	EXTRA KRUPUK	1000

Praktikum 2 | Menampilkan data dengan CROSS JOIN

1. Buatlah perintah untuk menampilkan data pelanggan nama dan data meja no, fasilitas dan ruang yang di pesan

	id_transaksi	id_karyawan	id_meja	id_pelangan	id_variasi	tanggal	metode_pembayaran	status_transaksi	id_karyawan	nama	alamat
1	TR01	KY04	M04P	PG01	V009	2022-01-04 00:00:00.000	CAST	SUKSES	KY04	SANDRA	PAJANGAN BANTUL
2	TR02	KY08	M04P	PG02	V009	2022-01-04 00:00:00.000	CAST	SUKSES	KY08	AYUNDA	WATES KULONPROGO
3	TR03	KY10	M05P	PG03	V003	2022-01-04 00:00:00.000	GO-PAY	SUKSES	KY10	BAGAS	PIYUNGAN WONOSARI
4	TR04	KY06	M05P	PG04	V009	2022-01-05 00:00:00.000	CAST	SUKSES	KY06	MAHARANI	WIROBRAJAN YOGYA
5	TR05	KY04	M05P	PG05	V009	2022-01-05 00:00:00.000	CAST	SUKSES	KY04	SANDRA	PAJANGAN BANTUL
6	TR06	KY12	M01P	PG06	V009	2022-01-06 00:00:00.000	GO-PAY	SUKSES	KY12	INDRA	RINGIN PUTIH SLEMAN
7	TR07	KY04	M01P	PG07	V009	2022-01-06 00:00:00.000	GO-PAY	SUKSES	KY04	SANDRA	PAJANGAN BANTUL
8	TR08	KY10	M04P	PG08	V001	2022-02-02 00:00:00.000	CAST	SUKSES	KY10	BAGAS	PIYUNGAN WONOSARI
9	TR09	KY06	M06P	PG09	V007	2022-02-02 00:00:00.000	CAST	SUKSES	KY06	MAHARANI	WIROBRAJAN YOGYA
10	TR10	KY04	M02P	PG10	V003	2022-02-03 00:00:00.000	GO-PAY	SUKSES	KY04	SANDRA	PAJANGAN BANTUL

2. Buatlah perintah untuk menampilkan data kategori, nama gerai, nama menu dan harga



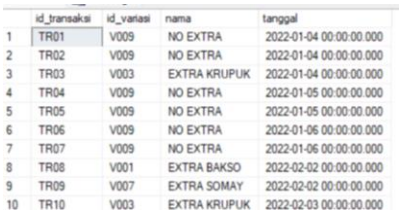
	kategori	nama	harga
1	OLAHAN AYAM	BAKSO	11000
2	OLAHAN AYAM	SOMAY	15000
3	OLAHAN AYAM	MIE AYAM	12000
4	OLAHAN AYAM	KUPAT TAHU	13500
5	OLAHAN AYAM	NASI GORENG	13000
6	OLAHAN AYAM	NASI PADANG	12000
7	OLAHAN AYAM	AYAM MADU 2	18000
8	OLAHAN AYAM	NASI SHOP	8000
9	OLAHAN AYAM	SATE	17000
10	OLAHAN AYAM	GORENGAN	5000

3. Buatlah perintah untuk menampilkan data pelanggan nama tanggal dan metode pembayaran yang bertransaksi di tanggal 2022-01-04

	nama	tanggal	metode_pembayaran	status_transaksi	no_hp
1	ANISA	2022-01-04 00:00:00.000	CAST	SUKSES	82235899147
2	NABILA	2022-01-04 00:00:00.000	CAST	SUKSES	89646599868
3	NEBULA	2022-01-04 00:00:00.000	GO-PAY	SUKSES	85156086291

Praktikum 5 | Menampilkan data dengan OUTER JOIN (FULL)

1. Buatlah perintah untuk menampilkan data transaksi dan viariasi, dengan tetap menampilkan data variasi yang tidak dipesan

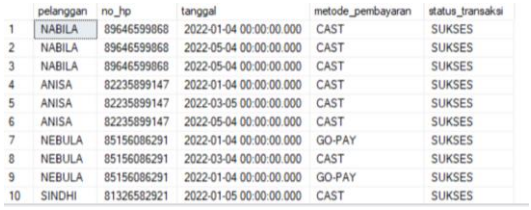


	id_transaksi	id_variasi	nama	tanggal
1	TR01	V009	NO EXTRA	2022-01-04 00:00:00.000
2	TR02	V009	NO EXTRA	2022-01-04 00:00:00.000
3	TR03	V003	EXTRA KRUPUK	2022-01-04 00:00:00.000
4	TR04	V009	NO EXTRA	2022-01-05 00:00:00.000
5	TR05	V009	NO EXTRA	2022-01-05 00:00:00.000
6	TR06	V009	NO EXTRA	2022-01-06 00:00:00.000
7	TR07	V009	NO EXTRA	2022-01-06 00:00:00.000
8	TR08	V001	EXTRA BAKSO	2022-02-02 00:00:00.000
9	TR09	V007	EXTRA SOMAY	2022-02-02 00:00:00.000
10	TR10	V003	EXTRA KRUPUK	2022-02-03 00:00:00.000

2. Buatlah perintah untuk menampilkan data agar daftar kategori dapat terlihat menu makanan dan harga

	id_kategori	nama	harga
1	KT09	GORENGAN	5000
2	KT10	KOPI	12500
3	KT10	JERUK	2000
4	KT10	TEH	2000
5	KT10	WEDANG UWOH	15000
6	KT10	JERUK	2000
7	KT10	JUS BUAH	12000
8	KT10	LEMONTIE	2500
9	KT10	MINUMAN KALENG	7500
10	KT10	LEMONTIE	2500

3. Buatlah perintah untuk menampilkan data transaksi



	pelanggan	no_hp	tanggal	metode_pembayaran	status_transaksi
1	NABILA	89646599868	2022-01-04 00:00:00.000	CAST	SUKSES
2	NABILA	89646599868	2022-05-04 00:00:00.000	CAST	SUKSES
3	NABILA	89646599868	2022-05-04 00:00:00.000	CAST	SUKSES
4	ANISA	82235899147	2022-01-04 00:00:00.000	CAST	SUKSES
5	ANISA	82235899147	2022-03-05 00:00:00.000	CAST	SUKSES
6	ANISA	82235899147	2022-05-04 00:00:00.000	CAST	SUKSES
7	NEBULA	85156086291	2022-01-04 00:00:00.000	GO-PAY	SUKSES
8	NEBULA	85156086291	2022-03-04 00:00:00.000	CAST	SUKSES
9	NEBULA	85156086291	2022-01-04 00:00:00.000	GO-PAY	SUKSES
10	SINDHI	81326582921	2022-01-05 00:00:00.000	CAST	SUKSES

Praktikum 6 | Menampilkan data dengan OUTER JOIN (RIGHT)

1. Buatlah perintah untuk menampilkan data menu yang tidak di-order

	id_menu	nama
1	MK05	NASI GORENG
2	MK08	NASI SHOP
3	MK15	JERUK
4	MK17	LEMONTIE
5	MK18	MINUMAN KALENG
6	MK20	TEH

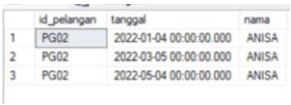
2. Buatlah perintah untuk menampilkan data pelanggan dan metode pembayaran berdasarkan transaksinya

	nama	metode_pembayaran
1	NABILA	CAST
2	ANISA	CAST
3	NEBULA	GO-PAY
4	SINDHI	CAST
5	DIMAS	CAST
6	RAFLY	GO-PAY
7	TEDDY	GO-PAY
8	MARIO	CAST
9	GHALY	CAST
10	KHARISMA	GO-PAY

3. Buatlah perintah untuk menampilkan data jenis makanan yang ada di gerai

	nama_gerai	jenis
1	BAKSO BAROKAH	Makanan
2	BAKSO BAROKAH	Minuman
3	PADANG JAYA	Makanan
4	PADANG JAYA	Minuman
5	MIE AYAM LARIS	Makanan
6	SEGO TAMBAH	Makanan
7	SEGO TAMBAH	Minuman
8	SOMAY MENTES	Makanan
9	NASGOS RAMINTEN	Makanan
10	NASGOS RAMINTEN	Minuman

4. Buatlah perintah untuk menampilkan data berapa banyak pelanggan Anisa bertransaksi



	id_pelangan	tanggal	nama
1	PG02	2022-01-04 00:00:00.000	ANISA
2	PG02	2022-03-05 00:00:00.000	ANISA
3	PG02	2022-05-04 00:00:00.000	ANISA

Praktikum 7 | Menampilkan data dengan OUTER JOIN (LEFT)

1. Buatlah perintah untuk menampilkan data pelanggan beserta berapa transaksinya

	id_pelangan	nama	jml_transaksi
1	PG01	NABILA	3
2	PG02	ANISA	3
3	PG03	NEBULA	3
4	PG04	SINDHI	6
5	PG05	DIMAS	1
6	PG06	RAFLY	1
7	PG07	TEDDY	8
8	PG08	MARIO	4
9	PG09	GHALY	1
10	PG10	KHARISMA	1

2. Buatlah perintah untuk menampilkan data nama makanan dan minuman berdasarkan gerainya

	nama	nama_gerai
1	BAKSO	BAKSO BAROKAH
2	TEH	BAKSO BAROKAH
3	NASI PADANG	PADANG JAYA
4	JERUK	PADANG JAYA
5	MIE AYAM	MIE AYAM LARIS
6	NASI SHOP	SEGO TAMBAH
7	LEMONTIE	SEGO TAMBAH
8	SOMAY	SOMAY MENTES
9	NASI GORENG	NASGOS RAMINTEN
10	TEH	NASGOS RAMINTEN

3. Buatlah perintah untuk menampilkan data metode pembayaran pada setiap meja yang dipesan

	no_meja	metode_pembayaran
1	MP02	GO-PAY
2	MP02	GO-PAY
3	MP02	CAST
4	MP02	GO-PAY
5	MP02	GO-PAY
6	MP02	CAST
7	MP02	GO-PAY
8	MP02	GO-PAY
9	MP02	CAST
10	MP04	GO-PAY

4. Buatlah perintah untuk menampilkan data menu dan kategori beserta harganya diurutkan dari bawah

	kategori	jenis	menu	harga
1	OLAHAN AYAM	Makanan	AYAM MADU 2	18000
2	OLAHAN SATE	Makanan	SATE	17000
3	OLAHAN SOMAY	Makanan	SOMAY	15000
4	MINUMAN	Minuman	WEDANG UWOH	15000
5	OLAHAN TAHU	Makanan	KUPAT TAHU	13500
6	OLAHAN NASI	Makanan	NASI GORENG	13000
7	MINUMAN	Minuman	KOPI	12500
8	MINUMAN	Minuman	JUS BUAH	12000
9	OLAHAN NASI	Makanan	NASI PADANG	12000
10	OLAHAN MIE	Makanan	MIE AYAM	12000

5. Buatlah perintah untuk menampilkan data menu dan harga yang dijual di setiap gerai

	nama_gerai	jenis	menu	harga
1	AYAM CEMANY MREBES	Makanan	AYAM MADU 2	18000
2	AYAM CEMANY MREBES	Minuman	LEMONTIE	2500
3	BAKSO BAROKAH	Minuman	TEH	2000
4	BAKSO BAROKAH	Makanan	BAKSO	11000
5	GORENG SRENG	Makanan	GORENGAN	5000
6	KEDAI SEGAR	Minuman	KOPI	12500
7	KEDAI SEGAR	Minuman	WEDANG UWOH	15000
8	KEDAI SEGAR	Minuman	JUS BUAH	12000
9	KEDAI SEGAR	Minuman	MINUMAN KALENG	7500
10	KUPAK TAHU BLALAK	Minuman	JERUK	2000

4.5 Tugas

1. Buat laporan praktikum dari percobaan yang dilakukan, masing-masing lengkapi dengan narasi, *query*, dan *screenshoot* hasil eksekusi. Tuliskan rangkuman dengan lengkap, runtut, dan mudah dipahami
2. Buat kesimpulan tentang apa yang anda dapatkan/pahami dari praktikum untuk materi **JOIN** yang baru dilakukan, sertakan kesimpulan tersebut dalam rangkuman anda

BAB V TRANSACTION

5.1 Tujuan

Setelah praktikum ini usai, mahasiswa diharapkan:

- Mampu menjelaskan perbedaan antara *autocommit*, *implicit transaction*, dan *explicit transaction* pada SQL Server
- mampu menunjukkan penggunaan *autocommit*, *implicit transaction*, dan *explicit transaction* pada SQL Server
- mmapu menggunakan semua metode transaksi pada SQL Server

5.2 Dasar Teori

Transaction atau transaksi adalah satu unit kerja. Jika transaksi berhasil, semua modifikasi data yang dilakukan selama transaksi dilakukan dan menjadi bagian permanen dari database. Jika transaksi mengalami kesalahan dan harus dibatalkan atau digulung balik (ROLLBACK), maka semua modifikasi data akan dihapus.

Properti ACID

Properti ACID di SQL Server memastikan integritas data selama transaksi. SQL ACID adalah singkatan dari *Atomicity*, *Consistency*, *Isolation*, *Durability*. Misalnya, pada sebuah transaksi transfer uang dari satu rekening ke rekening yang lainnya, jika terjadi sesuatu yang menyebabkan kegagalan transaksi ini, maka posisi saldo kedua rekening harus tetap pada kondisi sebelum transaksi dilakukan. Tidak boleh terjadi posisi di mana saldo rekening pengirim tidak terpotong, tetapi saldo rekening penerima sudah bertambah.

ATOMIC

Transaksi harus dijalankan tepat sekali dan harus atomik—baik semua pekerjaan selesai atau tidak sama sekali. Operasi dalam transaksi biasanya memiliki maksud yang sama dan saling diperlukan. Dengan hanya melakukan subset operasi ini, sistem dapat membahayakan niat

transaksi secara keseluruhan. Atomitas menghilangkan kemungkinan hanya memproses subset operasi.

CONCISTENCY

Transaksi harus mempertahankan konsistensi data, mengubah satu status data yang konsisten menjadi status data lain yang konsisten. Sebagian besar tanggung jawab untuk menjaga konsistensi jatuh ke pengembang aplikasi.

ISOLATION

Transaksi harus menjadi satuan isolasi, yang berarti bahwa transaksi bersamaan harus bertingkah seolah-olah masing-masing adalah satu-satunya transaksi yang berjalan dalam sistem. Karena isolasi tingkat tinggi dapat membatasi jumlah transaksi bersamaan, beberapa aplikasi mengurangi tingkat isolasi dengan imbalan *throughput* yang lebih baik.

DURABILITY

Transaksi harus dapat dipulihkan dan oleh karena itu harus memiliki durabilitas. Jika dilakukan *commit* terhadap sebuah transaksi, sistem akan menjamin bahwa pembaruannya dapat bertahan bahkan jika terjadi crash pada komputer, pembaruan ini berlaku segera setelah penerapan. *Logging* khusus yang memungkinkan prosedur hidupkan ulang sistem untuk menyelesaikan operasi yang belum selesai yang diperlukan oleh transaksi, membuat transaksi *durable*.

Mode Transaksi

Transaksi adalah sebuah unit kerja. Jika transaksi berhasil, maka semua modifikasi data yang terjadi selama transaksi dilakukan akan menjadi bagian permanen dari basisdata. Jika transaksi mengalami kesalahan dan harus dibatalkan atau *rollback*, maka semua modifikasi data akan dihapus dan dikembalikan ke status sebelumnya.

SQL Server beroperasi dalam mode transaksi berikut:

AUTOCOMMIT

Setiap pernyataan individu adalah transaksi.

EXPLICIT TRANSACTION

Setiap transaksi secara eksplisit dimulai dengan pernyataan BEGIN TRANSACTION dan secara eksplisit diakhiri dengan pernyataan COMMIT atau ROLLBACK.

IMPLICIT TRANSACTION

Transaksi baru secara implisit dimulai ketika transaksi sebelumnya selesai, tetapi setiap transaksi secara eksplisit diselesaikan dengan pernyataan COMMIT atau ROLLBACK.

SET IMPLICIT_TRANSACTIONS (SQL T-SQL)

Ketika ON, sistem berada dalam mode transaksi implisit. Ini berarti bahwa jika @@TRANCOUNT = 0, salah satu pernyataan Transact-SQL akan memulai transaksi baru.

Saat NONAKTIF, masing-masing pernyataan T-SQL sebelumnya dibatasi oleh BEGIN TRANSACTION yang tidak nampak dan pernyataan COMMIT TRANSACTION yang tidak diketahui.

```
DECLARE @IMPLICIT_TRANSACTIONS VARCHAR(3) = 'OFF';
IF ((2&@@OPTIONS)=2) SET @IMPLICIT_TRANSACTIONS = 'ON';
SELECT @IMPLICIT_TRANSACTIONS AS IMPLICIT_TRANSACTIONS;
```

5.3 Alat dan Bahan

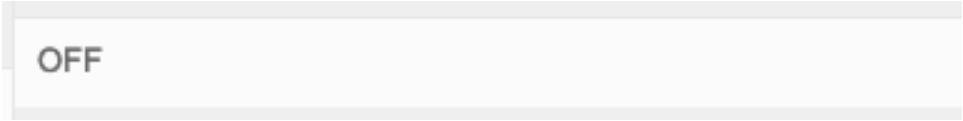
Database tools	SQL
File import	HR-sqlserver.sql dan HR-sqlserver-data.sql

5.4 Langkah Percobaan

Berikut ini adalah percobaan-percobaan yang akan kita lakukan:

Praktikum 1 | Melihat kondisi pengaturan IMPLICIT_TRANSACTIONS

Gunakan perintah select terhadap Global Variable @@OPTIONS untuk melihat pengaturan IMPLICIT_TRANSACTIONS



Praktikum 2 | Menjalankan perintah SQL dengan mode *autocommit*

Menggunakan ***mode autocommit***, lakukan perintah UPDATE terhadap tabel `countries`, SET `country_name` menjadi "Temporary Name" untuk `country_id` "AR".

country_id	country_name	region_id
AR	Temporary Name	2
AU	Australia	3
BE	Belgium	1

Praktikum 3 | Menjalankan perintah SQL dengan mode *implicit transactions*

Pertama, coba nyalakan mode IMPLICIT TRANSACTIONS dan jalankan perintah SQL tanpa memanggil perintah COMMIT atau ROLLBACK. Apa yang terjadi? Kenapa?

Kemudian, coba tambahkan perintah COMMIT/ROLLBACK TRANSACTION pada barisan kode perintah SQL. Apa yang terjadi?

country_id	country_name	region_id
AR	Argentina	2
AU	Australia	3
BE	Belgium	1

Praktikum 4 | Menjalankan perintah SQL dengan Mode *explicit transactions*

Sekarang kita akan mencoba mode *explicit transactions*. Jalankan perintah berikut, apa yang terjadi? Kenapa?

```
--set implicit_transactions OFF
--begin transaction
UPDATE countries SET country_name = 'Temporary Name' WHERE country_id = 'AR'
commit transaction
SELECT * FROM countries ORDER BY country_id
```

Kemudian coba jalankan perintah berikut, lengkap dengan klausa BEGIN TRANSACTION dan COMMIT/ROLLBACK TRANSACTION.

country_id	country_name	region_id
AR	Temporary Name	2
AU	Australia	3
BE	Belgium	1

5.5 Tugas

1. Buat laporan praktikum dari percobaan yang dilakukan, masing-masing lengkapi dengan narasi, *query*, dan *screenshoot* hasil eksekusi. Tuliskan rangkuman dengan lengkap, runut, dan mudah dipahami
2. Buat kesimpulan perihal apa yang anda dapatkan/pahami dari praktikum untuk materi **Transaction** yang baru dilakukan, sertakan kesimpulan tersebut dalam rangkuman anda

BAB VI T-SQL

6.1 Tujuan

Setelah praktikum ini usai, mahasiswa diharapkan:

- mengerti kapabilitas T-SQL pada SQL Server
- mampu membuat program sederhana menggunakan T-SQL
- mampu menggunakan DML di dalam T-SQL

6.2 Dasar Teori

Transact-SQL(T-SQL) menyediakan bahasa pemrograman yang kuat dengan fitur yang memungkinkan *user* untuk menyimpan nilai sementara dalam variabel, menerapkan eksekusi perintah bersyarat, meneruskan parameter ke prosedur tersimpan, dan mengontrol aliran program.

SQL adalah bahasa standar untuk mengakses dan memanipulasi basisdata. SQL menyediakan *procedural language* (bahasa prosedural) untuk memudahkan *user* dalam memanipulasi data sehingga user dimungkinkan seolah-olah berada dalam *environment* pemrograman. Bahasa prosedural ini ada pada beberapa vendor dengan penamaan yang berbeda: Transact-SQL (biasa disingkat T-SQL) di SQL Server, PL/SQL di Oracle, pgSQL di postgreSQL.

DEKLARASI VARIABEL

Pernyataan DECLARE menginisialisasi variabel T-SQL dengan:

- menentukan nama, nama variabel harus memiliki satu @ sebagai prefix atau karakter pertama
- menentukan tipe data beserta length atau panjang/ukuran data tersebut; pada variabel numerik, presisi dan skala juga ditetapkan; pada variabel jenis XML, koleksi skema opsional dapat ditetapkan
- mengatur nilai ke NULL

```
DECLARE @MyCounter INT;
```

Query di atas adalah perintah untuk membuat sebuah variabel dengan nama `@MyCounter` dan tipe data integer. Perintah `DECLARE` bisa digunakan untuk mendeklarasikan lebih dari satu variabel sekaligus. Berikut ini adalah contoh deklarasi *multiple variable*:

```
DECLARE @LastName NVARCHAR(30),
        @FirstName NVARCHAR(20),
        @StateProvince NCHAR(2);
```

Perintah di atas akan menghasilkan 3 buah variabel masing-masing:

- `@LastName` dengan tipe data NVARCHAR dengan panjang 30,
- `@FirstName` dengan tipe data NVARCHAR dengan panjang 20, dan
- `@StateProvince` dengan tipe data NCHAR dengan panjang 2.

OPERATOR PERBANDINGAN

Operator perbandingan menguji apakah dua ekspresi sama. Operator perbandingan dapat digunakan pada semua ekspresi kecuali ekspresi jenis data teks, ntext, atau gambar. Tabel berikut mencantumkan operator perbandingan Transact-SQL.

Tabel 6. 1 Operator Perbandingan Transact-SQL

Operator	Makna
= (sama dengan)	Sama dengan
> (lebih besar dari)	Lebih besar dari
< (kurang dari)	Kurang dari
>= (lebih besar atau sama dengan)	Lebih dari atau sama dengan
<= (kurang dari atau sama dengan)	Kurang dari atau sama dengan
<> (tidak sama dengan)	Tidak sama dengan
!= (tidak sama dengan)	Tidak sama dengan (bukan standar ISO)
!< (tidak kurang dari)	Tidak kurang dari (bukan standar ISO)
!> (tidak lebih besar dari)	Tidak lebih besar dari (bukan standar ISO)

PERCABANGAN IF ELSE

Memberlakukan kondisi pada eksekusi pernyataan T-SQL. Pernyataan Transact-SQL yang mengikuti kata kunci IF dan kondisinya dijalankan jika kondisi terpenuhi: ekspresi Boolean mengembalikan TRUE. Kata kunci ELSE opsional memperkenalkan pernyataan transact-SQL lain yang dijalankan ketika kondisi IF tidak terpenuhi: ekspresi boolean mengembalikan FALSE.

```
IF Boolean_expression
     { sql_statement | statement_block }
[ ELSE
     { sql_statement | statement_block } ]
```

6.3 Alat dan Bahan

Database tools	SQL
File import	HR-sqlserver.sql dan HR-sqlserver-data.sql

6.4 Langkah Percobaan

Berikut ini adalah percobaan-percobaan yang akan kita lakukan:

Praktikum 1 | Mendeklarasikan variabel, mengisikan nilai, dan menampilkannya

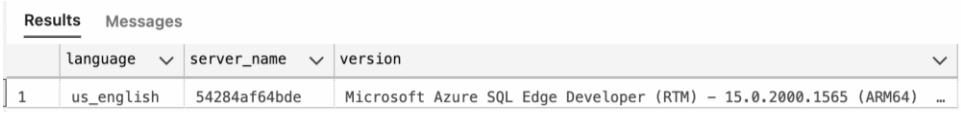
Deklarasikan variable @id dan @nama dengan tipe data INT dan VARCHAR(30).

Isi nilai 100 dan “Shofia” ke dalam variable @id dan @nama.

Cetak nilai dari variabel @id dan @nama pada layar console menggunakan klausa Print.

Praktikum 2 | Menampilkan nilai *global variable*

Tampilkan nilai dari *global variable* LANGUAGE, SERVERNAME, dan VERSION menggunakan perintah SELECT.

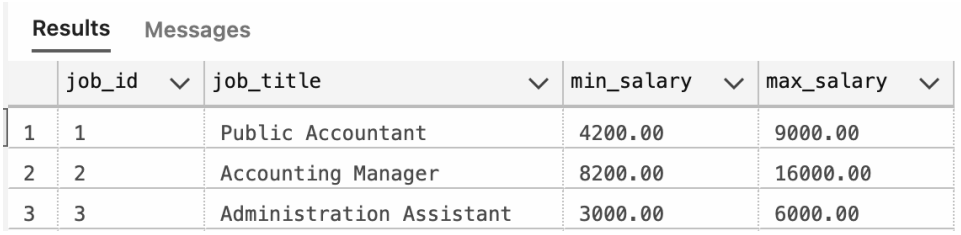


Results Messages

	language	server_name	version
1	us_english	54284af64bde	Microsoft Azure SQL Edge Developer (RTM) - 15.0.2000.1565 (ARM64) …

Praktikum 3 | Menggunakan T-SQL untuk menambahkan *record* pada tabel jobs

Cek terlebih dahulu seperti apa struktur tabel jobs dan contoh isinya, jika kita ingin menambahkan *record*/baris baru, nilai dan variabel apa saja yang kita butuhkan?



Results Messages

	job_id	job_title	min_salary	max_salary
1	1	Public Accountant	4200.00	9000.00
2	2	Accounting Manager	8200.00	16000.00
3	3	Administration Assistant	3000.00	6000.00

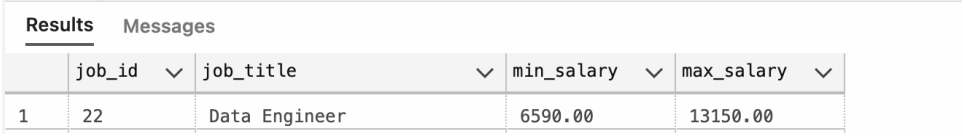
Tambahkan *record*/baris baru dengan memanfaatkan fungsi T-SQL, deklarasikan variabel-variabel yang dibutuhkan kemudian isikan nilainya

Results Messages

	job_id	job_title	min_salary	max_salary
1	22	Data Engineer	7000.00	12000.00

Praktikum 4 | Melakukan pembaharuan data dengan sebelumnya melakukan agregasi yang digunakan sebagai nilai perubahan

Hitung rata-rata gaji minimum dan gaji maksimum yang ada pada table jobs, gunakan hasil hitung tersebut untuk mengubah nilai min_salary dan max_salary dari job_title Data Engineer.

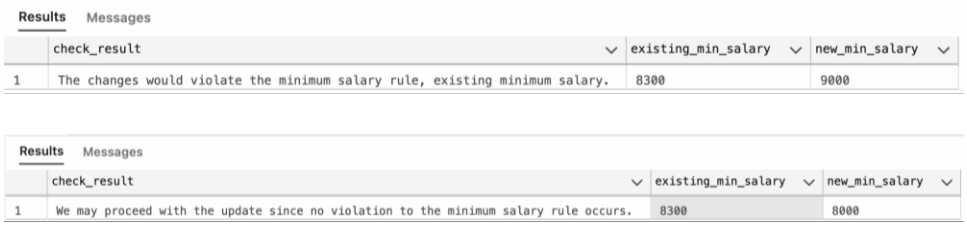


Results Messages

	job_id	job_title	min_salary	max_salary
1	22	Data Engineer	6590.00	13150.00

Praktikum 5 | Melakukan pengecekan data employees, apakah nilai min_salary baru akan melanggar aturan min_salary sebuah job_title tertentu

Kita ingin melakukan update `min_salary` pada sebuah `job_title`. Terlebih dahulu perlu dilakukan pengecekan pada data `employees`, jangan sampai perubahan `min_salary` melanggar aturan minimum gaji (contoh, kita memiliki data pegawai Public Accountant dengan gaji 8300, dilakukan perubahan `min_salary` pada tabel `jobs` dengan nominal 9000, hal ini tidak boleh dilakukan karena masih ada pegawai dengan nilai gaji di bawah 9000)



Results Messages

	check_result	existing_min_salary	new_min_salary
1	The changes would violate the minimum salary rule, existing minimum salary.	8300	9000

Results Messages

	check_result	existing_min_salary	new_min_salary
1	We may proceed with the update since no violation to the minimum salary rule occurs.	8300	8000

6.5 Tugas

1. Buat laporan praktikum dari percobaan yang dilakukan, masing-masing lengkapi dengan narasi, *query*, dan *screenshoot* hasil eksekusi. Tuliskan rangkuman dengan lengkap, runut, dan mudah dipahami
2. Buat kesimpulan perihal apa yang anda dapatkan/pahami dari praktikum untuk materi **T-SQL** yang baru dilakukan, sertakan kesimpulan tersebut dalam rangkuman anda.

BAB VII CURSOR

7.1 Tujuan

Setelah praktikum ini usai, mahasiswa diharapkan:

- mengetahui apa itu *cursor* dalam RDBMS
- mengerti kapabilitas dari *cursor*
- mampu menggunakan *cursor* untuk operasi RDBMS

7.2 Dasar Teori

Cursor dapat diartikan sebagai suatu variabel yang digunakan untuk menampung hasil *query* satu baris/*record* atau lebih, di mana ia juga menjadi penampung sekaligus pointer atas hasil eksekusi *query*. Cursor adalah sebuah tipe data yang dapat mengembalikan *value* berupa tabel.

Deklarasi Cursor

Cursor merupakan sebuah tipe data yang dapat dideklarasikan dengan cara sebagai berikut:

```
--declaration of variable
DECLARE @namadepan VARCHAR(30)
DECLARE @title VARCHAR(30)
--declaration of cursor data type, the name is employees_data
DECLARE employees_data CURSOR FOR
SELECT firstname,title FROM Employees
```

Mengakses data di dalam Cursor

Data di dalam Cursor ini kemudian bisa diakses dengan melakukan perintah OPEN dan FETCH sebagai berikut:

```
OPEN employees_data

--the position of cursor on the first record

FETCH NEXT FROM employees_data

INTO @namadepan, @title
```

Melakukan operasi terhadap data di dalam Cursor

Hasil dari operasi FETCH di atas dapat kita gunakan untuk operasi sesuai dengan kebutuhan. Cursor yang sudah selesai dipakai, perlu di-dealokasi dengan perintah DEALLOCATE untuk proses *release* memori yang digunakan.

```
WHILE @@FETCH_STATUS = 0

BEGIN

  --PRINT is the function that displays text and variable on the console
(screen).

  PRINT @namadepan +' -- '+@title

  --for each records/rows from employees table will be saved into
"@namadepan" and "@title" variable

  FETCH NEXT FROM employees_data

  INTO @namadepan, @title

END

--close the employees_data cursor

CLOSE employees_data

DEALLOCATE employees_data
```

7.3 Alat dan Bahan

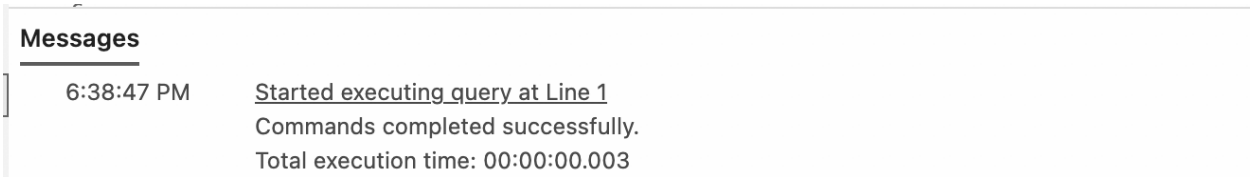
Database tools	SQL atau MySQL
File import	HR-sqlserver.sql dan HR-sqlserver-data.sql **atau** HR-mysql.sql dan HR-mysql-data.sql

7.4 Langkah Percobaan

Berikut ini adalah percobaan-percobaan yang akan kita lakukan:

Praktikum 1 | Mendeklarasikan Cursor dengan filter

Deklarasikan Cursor untuk kolom department_id dan department_name dari table departments dengan location_id 1700

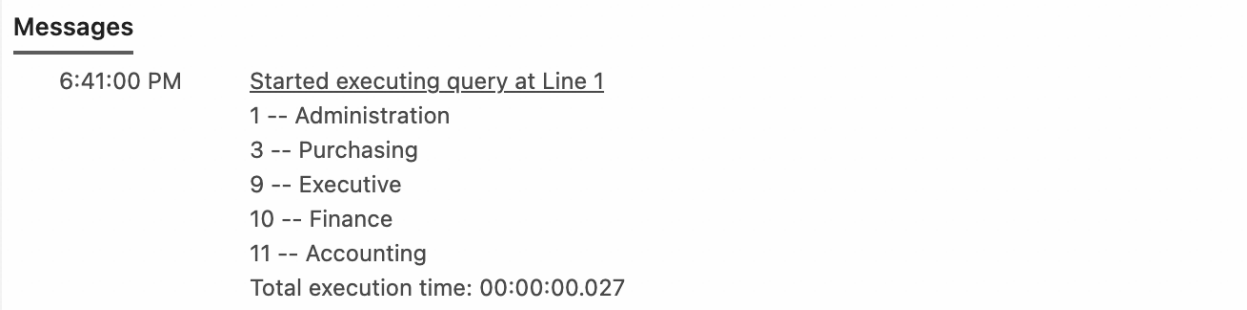


Praktikum 2 | Mengakses satu baris data dari cursor

Akses satu baris data dari Cursor departments yang sudah dibuat sebelumnya, print ke console.

Praktikum 3 | Melihat isi Cursor dari table departments (semua baris)

Akses semua baris data dari Cursor departments yang sudah dibuat sebelumnya, print ke console.



Praktikum 4 | Membuat latest_employees_cursor untuk data karyawan terbaru (satu tahun ke belakang dari maximum hire_date) dari table employees

Buat sebuah Cursor dari table employees untuk karyawan dengan hire_date <= 12 bulan dari hire_date karyawan terakhir (MAX).

Praktikum 5 | Melihat isi latest_employees_cursor (semua baris)

Akses semua baris data dari Cursor employees yang sudah dibuat sebelumnya, print ke console.

7.5 Tugas

1. Buat laporan praktikum dari percobaan yang dilakukan, masing-masing lengkapi dengan narasi, *query*, dan *screenshoot* hasil eksekusi. Tuliskan rangkuman dengan lengkap, runut, dan mudah dipahami
2. Buat kesimpulan perihal apa yang anda dapatkan/pahami dari praktikum untuk materi **Cursor** yang baru dilakukan, sertakan kesimpulan tersebut dalam rangkuman anda

BAB VIII FUNCTION dan STORED PROCEDURE

8.1 Tujuan

Setelah praktikum ini usai, mahasiswa diharapkan:

- mampu memahami konsep *function* dan *stored procedure*
- mampu membuat, menampilkan, dan menghapus *function*
- mampu membuat, menampilkan, dan menghapus *stored procedure*
- mampu membedakan format penulisan antara *function* dan *stored procedure*
- mampu menerapkan *function* dan *stored procedure* sesuai konteks yang tepat

8.2 Dasar Teori

FUNCTION

Function (fungsi) adalah sebuah program yang disimpan dalam sebuah basisdata yang bisa menerima parameter dan memberikan *return value*. Pada SQL ada dua macam *function* berdasarkan ketersediaannya yaitu ***built-in function*** dan ***user defined function***. *Built-in function* adalah fungsi bawaan yang sudah disediakan oleh SQL sehingga kita bisa menggunakannya tanpa membuat ulang, misalnya DATE(), ABS(), LENGTH(), dan lain-lain. Sedangkan *user-defined function* adalah fungsi yang dibuat sendiri oleh *user* biasanya merupakan program yang belum tersedia dalam bentuk *built-in function*.

Berdasarkan bentuk *return value*-nya, ada dua macam tipe *function* yaitu ***scalar function*** dan ***table-valued function***. *Scalar function* adalah fungsi yang mengembalikan *output* berupa nilai tunggal sesuai dengan tipe data yang didaftarkan pada klausa RETURNS. Tipe datanya apa saja kecuali TEXT, NTEXT, IMAGE, CURSOR, TIMESTAMP. Sementara *table-valued function* adalah fungsi yang mengembalikan *output* berupa *result set* (berbentuk tabel).

Keuntungan implementasi *function* antara lain: memungkinkan dibuat/dilakukannya *modular programming*, eksekusi yang lebih cepat, dan trafik dalam jaringan yang berkurang.

Ketika *user* membuat sebuah *user-defined function* baru, *function* tersebut akan tersimpan dalam metadata dan akan hilang ketika dihapus. *User* bisa melakukan modifikasi terhadap

function, misalnya melakukan perubahan terhadap nama *function*, perubahan baris kode, dan/atau menghapus *function* tersebut.

CREATE FUNCTION

Sintaks dasar:

```
CREATE FUNCTION function_name [ (parameter datatype [, parameter datatype])
]
RETURNS return_datatype
BEGIN
   declaration_section
   executable_section
END;
```

Penjelasan:

`function_name`

diisi dengan nama fungsi sesuai tujuan dari fungsi tersebut, misalnya `f_Insert_Customer_Data`

`parameter`

diisi dengan parameter sesuai kebutuhan fungsi tersebut, pendefinisian parameter diikuti dengan tipe datanya

`return_datatype`

tipe data dari *return value* yang diharapkan

`declaration_section`

bagian untuk mendeklarasikan *local variable* jika ada

`executable_section`

bagian yang berisi code dari *function* yang dibuat

Contoh (1) fungsi sederhana

```
CREATE FUNCTION f_Simple_Multiplication(var1 INT, var2 INT)
RETURNS INT
BEGIN
      RETURN (var1*var2)
END;
```

Contoh (2) dengan akses ke basisdata | *scalar function*

```
CREATE FUNCTION f_Count_Profit(productKey smallINT)
RETURNS INT
BEGIN
	DECLARE profit INT;
	SELECT profit=productsalescost-productactualcost
	FROM products
	WHERE productkey=productkey
	RETURN(profit)
END;
```

Contoh (3) dengan akses ke basisdata | *table-valued function*

```
CREATE FUNCTION f_Display_All_Product()
RETURNS TABLE
BEGIN
	RETURN (SELECT*FROM products)
END;
```

Catatan:

Ada beberapa perbedaan pembuatan *function* di MySQL dengan SQL Server, pada SQL Server setiap variabel harus diawali dengan prefix '@' sementara di MySQL tidak. Penulisan parameter dan deklarasi variabel pada SQL Server bisa menggunakan 'AS', contoh "`DECLARE @profit AS INT`"; sementara di MySQL tidak menggunakan 'AS'. 'AS' juga digunakan pada SQL Server sebelum klausa 'BEGIN'.

CALL FUNCTION

Function yang telah dibuat tidak serta merta langsung dieksekusi, cara memanggil *function* adalah dengan menggunakan klausa **SELECT**

```
SELECT function_name(param);
```

Untuk memanggil *function* yang baru saja dibuat, maka perintahnnya seperti ini:

```
SELECT f_Simple_Multiplication(20,15);

SELECT f_Count_Profit(1020);

SELECT*FROM f_Display_All_Product();
```

Kesalahan yang biasanya menimbulkan *error message* adalah karena nama *function* yang salah, jumlah parameter *input* yang berbeda entah lebih banyak atau lebih sedikit, atau adanya ketidaksesuaian tipe data pada parameter.

Jika dibutuhkan alias, maka penulisannya menjadi seperti ini:

```
SELECT f_Simple_Multiplication(20,15) AS multiplication_result;
```

SHOW LIST OF FUNCTION

Untuk menampilkan daftar *function* yang ada di basidata, sintaknya adalah sebagai berikut:

```
SHOW FUNCTION STATUS;
```

DROP FUNCTION

Jika sebuah *function* yang dibuat dirasa tidak diperlukan lagi, penghapusan bisa dilakukan terhadap *function* tersebut. Berikut ini adalah *basic syntax* untuk menghapus *function*:

```
DROP FUNCTION [ IF EXISTS ] function_name;
```

Contoh (1) menghapus `f_Simple_Multiplication` *function*

```
DROP FUNCTION f_Simple_Multiplication;
```

Perhatikan saat menghapus *function*, cukup nama *function*-nya saja yang dituliskan, tidak perlu dilengkapi dengan parameter. Penghapusan *function* hanya bisa dilakukan satu persatu, *multiple drop* tidak bisa dilakukan terhadap *function*.

STORED PROCEDURE

Stored procedure atau disebut juga dengan *procedure* adalah suatu program yang serupa dengan *function*, bisa menerima parameter tetapi tidak memiliki *return value* seperti pada *function*.

Keuntungan mengimplementasikan *stored procedure* antara lain: mengurangi trafik dalam jaringan, aman, code bersifat *reusable*, *maintenance* mudah, meningkatkan kinerja basisdata secara keseluruhan.

CREATE STORED PROCEDURE

Sintaks dasar:

```
CREATE PROCEDURE procedure_name [(parameter datatype [, parameter
datatype])]
BEGIN
   declaration_section
   executable_section
END;
```

Penjelasan:

`procedure_name`

diisi dengan nama *stored procedure*, biasanya diawali dengan prefix "sp_"

`parameter`

diisi dengan parameter, pada *stored procedure* ada 3 jenis parameter yaitu IN, OUT, dan INOUT

`declaration_section`

bagian untuk mendeklarasikan *local variable* jika ada

`executable_section`

bagian yang berisi code dari *stored procedure* yang dibuat

Contoh:

```
CREATE PROCEDURE sp_Display_CustomerData(customer_id varchar(20))
```

```
BEGIN
      SELECT * FROM customers
      WHERE customer_id=customer_id
END;
```

Stored procedure di atas bernama `sp_Display_CustomerData`, *stored procedure* ini menampilkan data customer berdasarkan customer_id-nya.

CALL STORED PROCEDURE

Stored procedure dipanggil/dieksekusi menggunakan klausa CALL, untuk memanggil *stored procedure* di atas penulisannya adalah adalah sebagai berikut:

```
CALL sp_Display_CustomerData("AE12345");
```

Pada SQL Server, *stored procedure* dieksekusi menggunakan klausa EXEC dan/atau EXECUTE

SHOW LIST OF STORED PROCEDURE

Untuk menampilkan dafar *function* yang ada di basidata, sintaknya adalah sebagai berikut:

```
SHOW PROCEDURE STATUS;
```

DROP STORED PROCEDURE

Jika *stored procedure* dirasa sudah tidak diperlukan, penghapusan bisa dilakukan terhadap *stored procedure* tersebut, *basic syntax*-nya adalah

```
DROP PROCEDURE [ IF EXISTS ] procedure_name;
```

Untuk menghapus *stored procedure* `sp_Display_CustomerData`, sintaksnya adalah

```
DROP PROCEDURE sp_Display_CustomerData;
```

8.3 Alat dan Bahan

MySQL atau SQL Server, import file .sql/.mdf berikut di *database tools* anda.

Database tools	MySQL
File import	product_managements.sql

8.4 Langkah Percobaan

Berikut ini adalah percobaan-percobaan yang akan:

Praktikum 1 | Function

1. Buat sebuah *function* **f_LuasPermukaanKubus**, fungsi ini digunakan untuk menghitung luas permukaan kubus, L = 6*sisi*sisi. Fungsi ini membutuhkan satu buah parameter input untuk memasukkan nilai sisi kubus. Suggestion: manfaatkan *math built-in function*

   Setelah fungsi berhasil dibuat, lakukan beberapa percobaan eksekusi dengan mengganti nilai parameternya.

   Output:

f_LuasPermukaanKubus(2)
24

2. Buat sebuah *function* **f_VolumePersegiPanjang**, fungsi ini digunakan untuk menghitung volume persegi panjang, V = panjang*lebar*tinggi. Fungsi ini membutuhkan 3 buah parameter input untuk memasukkan nilai panjang, lebar, dan tinggi.

   Setelah fungsi berhasil dibuat, lakukan beberapa percobaan eksekusi dengan mengganti nilai parameternya.

   Contoh output:

f_VolumePersegiPanjang(5,5,1)
25

3. Tampilkan daftar *function* yang sudah anda buat, *seharusnya ada dua buah function pada daftar tersebut*

Db	Name	Type
product_managements	f_LuasPermukaanKubus	FUNCTION
product_managements	f_VolumePersegiPanjang	FUNCTION

4. Buat sebuah *function* untuk menentukan kategori nilai, dengan ketentuan:

   A 91 – 100, B 71 – 90, C 51 – 70, D kurang dari 50

   Fungsi ini dinamai **f_KategoriNilai**, dengan satu buah parameter *input* untuk memasukkan nilai, return berupa char.

   Contoh output:

f_KategoriNilai(49)
D

5. Hapus *function* dengan nama **f_VolumePersegiPanjang**
6. Tampilkan kembali daftar *function* di basis data anda, *seharusnya function yang dihapus tidak ditemukan saat daftar function ditampilkan*

Praktikum 2 | *Stored Procedure*

1. Buat sebuah *stored procedure* (sp) dengan nama `sp_products_limit_3`, sp ini tidak memiliki parameter input, sp ini berisi perintah untuk menampilkan data dari tabel `products`, data yang ditampilkan adalah keseluruhan kolom dan 3 buah baris data teratas saja.
2. Eksekusi `sp_products_limit_3!` Tuliskan kode eksekusinya, dan *screenshoot* hasil eksekusi sp tersebut.
3. Buah sebuah sp dengna nama `sp_insert_new_concerns`, sp ini menerima satu buah parameter dengan nama: name, tipe data varchar(20); ketika sp dieksekusi, maka akan ada tambahan satu baris data baru di tabel concern, dengan nilai sesuai input yang dimasukkan saat sp dieksekusi.

   Contoh eksekusi: call `sp_insert_new_concerns("dark under eye")`
4. Eksekusi `sp_insert_new_concerns!` Tuliskan kode eksekusinya, dan *screenshoot* hasil eksekusi sp tersebut.
5. Tampilkan dua buah *stored procedure* yang baru dibuat.
6. Hapus sp dengan nama `sp_products_limit_3!`
7. Buktikan bahwa sp tersebut benar sudah terhapus (tampilkan daftar sp terbaru).

8.5 Tugas

8. Buat laporan praktikum dari percobaan yang dilakukan, masing-masing lengkapi dengan narasi, *query*, dan *screenshoot* hasil eksekusi. Tuliskan rangkuman dengan lengkap, runut, dan mudah dipahami
9. Buat kesimpulan perihal apa yang anda dapatkan/pahami dari praktikum untuk materi **Function dan Stored Procedure** yang baru dilakukan, sertakan kesimpulan tersebut dalam rangkuman anda

BAB IX TRIGGER

9.1 Tujuan

Setelah praktikum ini usai, mahasiswa diharapkan:

- mampu memahami konsep trigger
- mampu membuat, menampilkan, dan menghapus trigger
- mampu mengimplementasikan penggunaan trigger sesuai pada konteks yang tepat

9.2 Dasar Teori

TRIGGER

Trigger adalah *stored procedure* yang dieksekusi secara otomatis ketika sebuah *event* terjadi dalam server basisdata. Biasanya event ini terjadi ketika *user* ingin melakukan perubahan data dengan Data Manipulation Language (DML). Trigger bisa terjadi pada DML maupun pada Data Definition Language (DDL).

Perbedaan antara trigger dan *stored procedure*:

Trigger baru terjadi jika ada *event* yang memicu, sementarara *stored procedure* terjadi jika dieksekusi menggunakan klausa EXEC dan/atau CALL. Trigger tidak bisa memiliki parameter sementara *stored procedure* bisa memiliki satu atau lebih parameter.

Ada 3 macam TRIGGER berdasarkan operasi DML yang dilakukan:

- INSERT TRIGGER adalah trigger yang akan dijalankan ketika ada operasi penambahan baris data pada tabel
- UPDATE TRIGGER adalah trigger yang akan dijalankan ketika ada operasi pengubahan baris data pada tabel
- DELETE TRIGGER adalah trigger yang akan dijalankan ketika ada operasi penghapusan baris data pada sebuah tabel.

CREATE TRIGGER

Berikut ini adalah sintaks dasar untuk membuat trigger:

```
CREATE TRIGGER trigger_name

{BEFORE | AFTER} {INSERT | UPDATE| DELETE }

ON table_name FOR EACH ROW

trigger_body;
```

Penjelasan:

`trigger_name`

nama trigger, nama trigger sebaiknya dituliskan secara informatif sesuai dengan tujuan trigger

`BEFORE | AFTER`

trigger time, BEFORE berarti bahwa trigger akan dieksekusi sebelum perubahan dibuat permanen dalam basisdata; AFTER berarti bahwa trigger akan dieksekusi setelah perubahan dibuat permanen dalam basisdata

`INSERT | UPDATE| DELETE`

trigger event adalah *event* yang didaftarkan dalam pembuatan trigger, *event* bisa berUpa *single event* (salah satu dari pilihan *event* tersebut di atas) atau *multiple event* (dua atau lebih event yang merupakan kombinasi dari event tersebut di atas); penulisan *multiple event* dipisahkan menggunakan koma

`table_name`

nama tabel yang didaftarkan pada trigger

`trigger_body`

isi dari trigger

Contoh (1) membuat sebuah after trigger dengan event UPDATE

```
CREATE TRIGGER after_update_concern

AFTER UPDATE

ON concerns FOR EACH ROW

    INSERT INTO concern_log (concern_id, last_name, date_changed)

    VALUES (OLD.concern_id, OLD.name, NOW());
```

Ketika sebuah *event* UPDATE terjadi pada tabel concerns, maka secara otomatis perintah INSERT ke tabel concern_log akan terjadi. Ketika UPDATE sudah selesai dan perubahan sudah permanen dalam tabel, akan terjadi pemasukkan data ke tabel concern_log.

Contoh (2) membuat sebuah before trigger dengan event DELETE

```
CREATE TRIGGER before_delete_review
BEFORE DELETE

ON reviews FOR EACH ROW

       INSERT INTO review_log (review_id, customer_id, content,
date_deleted)

       VALUES (OLD.review_id, OLD.customer_id, OLD.content, NOW());
```

Ketika sebuah event UPDATE terjadi pada tabel re*view*s, maka secara otomatis akan dijalankan perintah INSERT ke tabel re*view*_log.

LIST TRIGGER

Berikut ini adalah perintah untuk menampilkan trigger:

```
SHOW TRIGGERS;
```

Ketika perintah di atas dieksekusi, maka informasi dari seluruh trigger yang ada dalam server basisdata anda akan ditampilkan. Informasi tersebut berisi antara lain: nama trigger, *trigger event*, tabel yang didaftarkan pada trigger, *statement* atau isi dari trigger, dan trigger time.

Untuk menampilkan daftar trigger di basisdata tertentu,

```
SHOW TRIGGERS {FROM|IN} database_name;
```

Ganti `database_name` sesuai basisdata yang anda inginkan.

Contoh:

```
SHOW TRIGGERS FROM selling;
```

Perintah di atas akan menampilkan seluruh trigger yang ada pada basisdata dengan nama selling;

DROP TRIGGER

Jika trigger dirasa sudah tidak diperlukan dan ingin dilakukan penghapusan trigger dari basisdata, sintaknya sebagai berikut:

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name;
```

User dimungkinkan untuk melakukan *multiple drop* pada trigger, cukup dengan mendaftarkan nama trigger yang ingin dihapus dan dipisahkan dengan koma antara satu trigger dengan yang lain.

Contoh (1) *single drop*

```
DROP TRIGGER update_product;
```

Perintah di atas akan menghapus satu buah trigger dengan nama `update_product`.

Contoh (2) *multiple drop*

```
DROP TRIGGER update_product, delete_customer;
```

Perintah di atas akan menghapus dua buah trigger dengan nama `update_product` dan `delete_customer`.

9.3 Alat dan Bahan

Database tools	MySQL

File import	product_managements.sql

9.4 Langkah Percobaan

Berikut ini adalah percobaan-percobaan yang akan:

Praktikum 1 | Membuat log aktivitas berdasarkan *update* yang terjadi dalam tabel

Eksekusi perintah berikut:

1. Buat sebuah tabel dengan nama `log_skin_concern`, dengan kolom `log_id`, `concern_id`, `last_name`, `last_update`
2. Buat sebuah AFTER trigger dengan event UPDATE pada tabel `concerns`. Ketika sebuah UPDATE statement terjadi di tabel `concerns` maka ada sebuah perintah yang otomatis akan dieksekusi, perintah tersebut adalah INSERT statement ke tabel `log_skin_concern`, ketentuan *value* yang dimasukkan adalah sebagai berikut:

   `concern_id`: sesuai dengan concern_id dari baris yang dilakukan UPDATE

   `last_name`: sesuai dengan OLD value dari name

   `last_update`: sesuai dengan waktu eksekusi dilakukan

3. Setelah trigger dibuat, tampilkan daftar triggernya menggunakan perintah **`SHOW TRIGGER`**
4. Eksekusi sebuah perintah untuk membangkitkan trigger, dalam hal ini UPDATE statement di tabel **`concerns`**. Kemudian lihat apakah perubahan terjadi di tabel **`concerns`**, dan lihat pula apakah ada data yang otomatis dimasukkan ke tabel **`log_skin_concern`**

9.5 Tugas

1. Buat laporan praktikum dari percobaan yang dilakukan, masing-masing lengkapi dengan narasi, *query*, dan *screenshoot* hasil eksekusi. Tuliskan rangkuman dengan lengkap, runut, dan mudah dipahami
2. Buat kesimpulan perihal apa yang anda dapatkan/pahami dari praktikum untuk materi **Trigger** yang baru dilakukan, sertakan kesimpulan tersebut dalam rangkuman anda.

BAB X INDEX

10.1 Tujuan

Setelah praktikum ini usai, mahasiswa diharapkan:

- mampu memahami konsep index
- mampu membuat, menampilkan, dan menghapus index
- mampu mengimplementasikan penggunaan index pada konteks yang tepat

10.2 Dasar Teori

INDEX

Ketika anda membaca buku, anda bisa temukan index di bagian belakang buku. Index pada buku berupa kosakata dan nomor halaman yang dituliskan secara berurutan dari A ke Z. Untuk mendapatkan/mencapai kata tertentu, ketimbang mencari ke halaman buku satu persatu, menggunakan informasi pada index akan membuat pencarian menjadi lebih cepat. Index pada SQL juga tidak berbeda dengan index pada buku.

Index adalah urutan nilai yang masing-masing nilainya memiliki petunjuk atau pointer di mana nilai itu disimpan. Index adalah sebuah struktur data yang menyediakan kecepatan akses terhadap baris dalam sebuah tabel berdasarkan nilai dari satu atau lebih kolom. Kecepatan akses pada penggunaan index ini dapat meningkatkan kinerja pemrosesan *query*. Index memungkinkan SQL untuk mendapatkan hasil pencarian tanpa harus melakukan *scanning* satu persatu ke dalam tabel. Tetapi secara bersamaan penggunaan index dapat meningkatkan penggunaan memori yakni 2-3 kali lebih besar.

CREATE INDEX

Berikut ini adalah sintaks dasar untuk membuat index:

```
CREATE [UNIQUE] INDEX index_name ON table_name (columnName [ASC|DESC][,…])
```

Penjelasan:

`UNIQUE`

UNIQUE bersifat opsional, penambahan klausa UNIQUE pada saat pembuatan index akan memaksakan nilai unik pada kolom yang didaftarkan sebagai *index key*. Sebaliknya, tanpa penambahan UNIQUE, *index key* dibolehkan untuk memiliki nilai duplikat. UNIQUE hanya bisa diterapkan terhadap kolom yang memiliki *constraint* UNIQUE/PRIMARY KEY

`index_name`

nama index

`table_name`

nama table yang akan dibuat index-nya.

Contoh (1) membuat index

```
CREATE UNIQUE INDEX idx_customer_id ON customers (customer_id);
```

Contoh (2) membuat index dengan detail ASC/DESC

```
CREATE UNIQUE INDEX idx_item_id ON items (item_id DESC);
```

Contoh (3) membuat index saat *creating table*

```
CREATE TABLE t (
     t_id   INT PRIMARY KEY,
     c1 INT,
     c2 INT,
     INDEX(c1));
```

Perintah di atas akan menghasilkan:

- satu buah tabel dengan nama ‘t’,
- dua buah index masing-masing adalah
    - satu *clustered index* dengan nama ‘PRIMARY’ dengan *index key* dari kolom primary key yaitu t_id bersifat **UNIQUE**

- satu *non-clustered index* dengan nama c1 dengan *index key* dari kolom c1 bersifat **tidak UNIQUE**

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null
PRIMARY	BTREE	Yes	No	t_id	0	A	No
c1	BTREE	No	No	c1	0	A	Yes

Gambar 8. 1 Daftar index hasil eksekusi perintah contoh (3)

Contoh (4) membuat index dengan ALTER TABLE

```
ALTER TABLE t ADD INDEX(c2);
```

Hasil eksekusi ALTER TABLE di atas adalah satu buah index di table 't' dengan *index key* dari kolom c2 bersifat tidak UNIQUE.

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null
PRIMARY	BTREE	Yes	No	t_id	0	A	No
c1	BTREE	No	No	c1	0	A	Yes
c2	BTREE	No	No	c2	0	A	Yes

Gambar 8. 2 Tambahan index c2 di tabel 't'

SHOW INDEX

Untuk menampilkan index dari tabel tertentu:

```
SHOW INDEXES FROM table_name;
```

Untuk menampilkan index dari tabel di basisdata tertentu:

```
SHOW INDEXES FROM table_name IN database_name;
atau,
SHOW INDEXES FROM database_name.table_name;
```

Perintah di atas akan menampilkan *result-set* yang sama.

DROP INDEX

Sintaks dasar:

```
DROP INDEX index_name ON table_name;
```

Contoh:

```
DROP INDEX c1 ON t;
```

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null
PRIMARY	BTREE	Yes	No	t_id	0	A	No
c2	BTREE	No	No	c2	0	A	Yes

Gambar 8. 3 Daftar index di tabel 't' setelah c1 dihapus

10.3 Alat dan Bahan

Database tools	MySQL
File import	product_managements.sql

10.4 Langkah Percobaan

Berikut ini adalah percobaan-percobaan yang akan:

Praktikum 1 | Membuat index dengan beberapa cara, melihat daftar index pada tabel, menghapus index

1. Buat sebuah tabel dengan nama t1,
    - T_id INT PRIMARY KEY
    - C1 varchar(20)
    - C2 INT
    - C3 INT

    Tambahkan index dengan nilai *index key* berasal dari kolom C1 saat pembuatan tabel
2. Tampilkan daftar index di tabel t1

3. Menggunakan **ALTER TABLE statement**, buat sebuah ***composit index*** di tabel t1 dengan *index key* dari kolom C1 dan C2
4. Tampilkan daftar index di tabel t1, *seharusnya ada penambahan satu buah index di tabel t1 sehingga total index adalah 3 buah*.
5. Menggunakan **CREATE INDEX statement**, buah sebuah index di tabel t1 dengan *index key* C3, dan *key name* idx_c3
6. Tampilkan daftar index di tabel t1, *seharusnya ada penambahan satu buah index di tabel t1 sehingga total index adalah 4 buah*
7. Hapus index dengan nama idx_c3
8. Tampilkan kembali daftar index untuk membuktikan index idx_c3 benar terhapus

Praktikum 2 | Melihat perbedaan kecepatan akses antara tabel dengan dan tanpa index

1. Eksekusi perintah berikut:

```
SELECT*FROM customers WHERE join_date >= '2022-07-04';
```

Perhatikan *query execution time* yang dibutuhkan untuk mengeksekusi perintah di atas. Kemudian, eksekusi perintah berikut ini:

```
EXPLAIN SELECT*FROM customers WHERE join_date = '2022-07-04';
```

Perhatikan angka di kolom `rows`. Angka tersebut menunjukkan jumlah baris yang di-*scan* selama pencarian

2. Buat sebuah index untuk tabel `customers` dengan nama `idx_cust_join_date`, *index key* dari kolom `join_date`
3. Eksekusi kembali perintah nomor 1. Perhatikan *query execution time* dan angka di kolom `rows`, kemudian bandingkan angka tersebut dengan angka di percobaan sebelumnya.

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	customers	ALL	*NULL*	*NULL*	*NULL*	*NULL*	10	Using where

10.5 Tugas

1. Buat laporan praktikum dari percobaan yang dilakukan, masing-masing lengkapi dengan narasi, *query*, dan *screenshoot* hasil eksekusi. Tuliskan rangkuman dengan lengkap, runut, dan mudah dipahami
2. Buat kesimpulan perihal apa yang anda dapatkan/pahami dari praktikum untuk materi **Index** yang baru dilakukan, sertakan kesimpulan tersebut dalam rangkuman anda.

BAB XI VIEW

11.1 Tujuan

Setelah praktikum ini usai, mahasiswa diharapkan:

- mampu memahami konsep serta penggunaan *view* dalam basis data
- memahami memahami penerapan konsep, pembuatan dan modifikasi, *view*.
- mampu menyelesaikan kasus-kasus pengambilan data dengan menggunakan pendekatan *view*.

11.2 Dasar Teori

View merupakan *virtual table* yang dibangun dari satu atau beberapa tabel. *View* tidak membuat penyimpanan secara fisik seperti tabel namun hanya menyimpan referensi/*pointer* ke *record* pada tabel yang berkaitan. Selain menampilkan hasil, *view* juga dapat digunakan sebagai sumber data saat perintah dijalankan, *view* juga berguna untuk membatasi akses basis data, membuat perintah kompleks secara mudah, mengizinkan independensi data dan untuk menampilkan bentuk data yang berbeda dari data yang sama. Apa yang tersimpan dalam *view* hanya perintah seleksi data, namun bentuk fisik *view* dapat diperlakukan sebagaimana sebuah tabel.

Keuntungan view di antaranya:

1. Lebih ringkas dalam penulisan perintah *query* karena sudah terwakili oleh *view*, sehingga penulisan *query* menjadi sederhana
2. Demi keamanan, karena *user* tidak secara langsung mengakses tabel secara fisik
3. Kompleksitas pengambilan data dapat disembunyikan dalam *view*
4. *User* hanya perlu tahu nama dari *view*
5. Data yang direferensi oleh *view* dapat juga dilakukan SELECT dengan perintah lainnya dan hasilnya kemudian di-*filter*

Sintaks dasar:

```
CREATE VIEW [nama_view] ([daftar_field])
AS
    [select_statement];
```

Contoh (1) *view* dengan SELECT statement yang mengandung satu buah JOIN

```
CREATE VIEW v_gerai
AS
    SELECT gerai.nama_gerai, nama FROM menu
    RIGHT JOIN gerai ON menu.id_gerai=gerai.id_gerai;
```

Contoh (2) *view* dengan SELECT statement yang mengandung join dengan 2 tabel

```
CREATE VIEW v_gerai_detail
AS
     SELECT gerai.nama_gerai, kategori.kategori, menu.jenis, menu.nama,
     menu.harga FROM gerai
     LEFT JOIN menu ON gerai.id_gerai = menu.id_gerai
     LEFT JOIN kategori ON menu.id_kategori = kategori.id_kategori;
```

Perintah di atas merupakan perintah untuk membuat *view* `v_gerai` dan `v_gerai_detail` dimana perintah tersebut akan digunakan untuk menampilkan beberapa *record* dari beberapa tabel yang sering dibutuhkan dalam pencarian informasi terkait dengan basis data tertentu.

View yang dibuat akan tersimpan dalam *database engine*, untuk memanggil atau mengeksekusi *view* yang sudah dibuat cukup memanggil *view* seperti halnya melakukan SELECT statement terhadap tabel.

```
SELECT * FROM name_of_view;
```

Sebagaimana SELECT statement pada umumnya, saat SELECT terhadap *view*, *user* dapat menambahkan klausa WHERE dan detail SELECT statement lain jika dibutuhkan.

Ada tiga jenis view, antara lain: *horizontal view, vertical view*, dan *view on view*. Berikut paparannya:

Horizontal View

Horizontal view membatasi akses pengguna terhadap satu atau lebih baris dari tabel yang dipilih. Pada dasarnya *horizontal view* lebih menekankan pada hasil *record* pada baris yang dipilih.

Contoh (3) *horizontal view*

```
CREATE VIEW v_nasabah
AS
    SELECT * FROM nasabah WHERE kd_cabang='KC001';
```

	Kd_nasabah	Nama	Alamat	Rekening	Kd_cabang	Saldo
1	N0005	Memey	Seyegan	12340005	KC001	1200000
2	N0007	Julia	Monjali	12340007	KC001	1400000
3	N0008	Agusto	Turi	12340008	KC001	2200000

Gambar 11. 1 Hasil eksekusi v_nasabah

Vertical View

Vertical view membatasi akses pengguna ke kolom yang dipilih dari satu atau lebih. sama halnya dengan *horizontal view* akan tetapi pada *vertical view* ini lebih menekankan hasil *record* pada kolom yang dipilih.

Contoh (4) *vertical view*

```
CREATE VIEW v_nasabah_new
AS
    SELECT kd_nasabah,nama,alamat,rekening,saldo FROM nasabah
    WHERE kd_cabang='kc001';
```

	Kd_nasabah	Nama	Alamat	Rekening	Saldo
1	N0005	Memey	Seyegan	12340005	1200000
2	N0007	Julia	Monjali	12340007	1400000
3	N0008	Agusto	Turi	12340008	2200000

Gambar 11. 2 Hasil eksekusi v_nasabah_new

View On View

View on view artinya *view* yang dibuat berdasarkan *view* yang sudah dibuat sebelumnya. Pada *view on view* terdapat istilah *parent* dan *child*, *parent* adalah *view* yang menjadi rujukan saat sebuah *view* lain dibuat, *child* adalah *view* yang dibuat berdasarkan *view* sebelumnya yang sudah ada terlebih dahulu.

Contoh (5) *view on view*

```
CREATE VIEW v_nasabah_v
AS
SELECT kd_nasabah, nama, alamat, rekening, saldo FROM v_nasabah;
```

Penamaan view, baik itu horizontal view, vertical view, maupun view on view, tidak perlu diberi kata yang menunjukkan bahwa view yang dibuat adalah jenis view tertentu, contohnya nama yang tidak perlu: v_horizontal_nasabah_jakarta. Penamaan view sebaiknya diawali dengan prefix v_, ini guna memudahkan membedakan view dengan objek basisdata yang lain.

Modifikasi *view*

Seperti halnya sebuah objek basis data, *view* juga bisa di-*update* dengan perintah `ALTER VIEW`. Pada dasarnya, jika basis data mampu menentukan pemetaan balik dari skema *view* ke skema tabel dasar, maka *view* memungkinkan untuk di-*update*. Dalam kondisi ini, operasi yang dapat digunakan yakni *INSERT, UPDATE* dan *DELETE* dapat diterapkan pada *view*. Adapun perintah sebagai berikut:

```
ALTER VIEW [nama_view]
AS
    SELECT[fiedl_1],[fiedl_2],[…],  [fiedl_n]
    FROM [table_1],[table_2,[…],[table_n]
    WHERE [table_1] ([key]) = [table_2] ([key]);
```

Contoh sintak merubah *view* sebagai berikut :

```
ALTER VIEW v_gerai
AS
    SELECT kategori.kategori, menu.jenis, menu.nama, menu.harga
    FROM menu LEFT JOIN kategori ON menu.id_kategori = kategori.id_kategori;
```

```
ALTER VIEW v_gerai_detail
AS
    SELECT gerai.nama_gerai, kategori.kategori, menu.jenis, menu.nama,
    menu.harga FROM gerai
    LEFT JOIN menu ON gerai.id_gerai = menu.id_gerai
    LEFT JOIN kategori ON menu.id_kategori = kategori.id_kategori
    WHERE jenis = 'Minuman';
```

Perintah di atas memberikan gambaran bahwa pada saat *view* sudah ada dan butuh tambahan informasi lebih, kita dapat merubah pada *select statement*nya.

Untuk memanggil atau mengakses data *view* dapat digunakan perintah *Select* sama halnya ketika ingin mengakses sebuah basis data hanya saja untuk mengakses data pada *view* ini cukup dengan memanggil nama *view* saja. Hal ini yang akan mempercepat proses tampilan data Adapun beberapa perintah yang dapat digunakan sebagai berikut:

```
SELECT fiedl_1, fiedl_2, fiedl_n
FROM nama_view
      [WHERE condition]
      ORDER BY [fiedl_name,……,……] Type;
```

```
SELECT * FROM v_gerai;
```

```
SELECT * FROM v_gerai WHERE jenis = 'Minuman''
```

```
SELECT * FROM v_gerai_detail WHERE harga > 10000;
```

```
SELECT nama_gerai,jenis,harga FROM v_gerai_detail WHERE nama_gerai LIKE
'_a%';
```

Sedangkan untuk mengubah nama *view*, dapat kita gunakan perintah sebagai berikut:

```
EXEC sp_rename
      @objname = 'nama view lama'
      @newname = 'nama view baru';
```

Sehingga apabila sewaktu waktu ada perubahan nama *view* atau ada nama *view* yang sama bisa dilakukan perubahan **.**

Contoh :

```
EXEC sp_rename
      @objname = 'v_penjualan'
      @newname = 'v_laporan_penjualan';
```

Sedangkan untuk menghapus *VIEW* yang sudah tidak diperlukan, dapat kita gunakan perintah sebagai berikut:

```
DROP VIEW [nama_view];
```

11.3 Alat dan Bahan

Database tools	SQL atau MySQL
File import	FoodsCenter.sql/.mdf

11.4 Langkah Percobaan

Berikut ini adalah percobaan-percobaan yang akan:

Praktikum 1 | Membuat, Menampilkan, Mengubah dan Menghapus *View*

1. Buatlah perintah untuk membuat *view* dengan nama v_invoice untuk menampilkan id transaksi, nama menu, harga menu, jumlah yang dibeli, total harga dan tanggal transaksi. Tampilkan *view* dengan urutan tanggal secara *ascending*.
2. Buatlah perintah untuk merubah *view* v_invoice dengan menambahkan informasi kasir dan ubalah *view* tersebut menjadi v_invoices. Tampilkan dengan urutan tanggal secara *ascending*.
3. Buatlah sebuah horizontal *view* dari basisdata yang anda gunakan.

4. Buatlah sebuah vertical *view* dari basisdata yang anda gunakan
5. Buatlah sebuah *view* on *view* dari basisdata yang anda gunakan.

11.5 Tugas

1. Buat laporan praktikum dari percobaan yang dilakukan, masing-masing lengkapi dengan narasi, *query*, dan *screenshoot* hasil eksekusi. Tuliskan rangkuman dengan lengkap, runut, dan mudah dipahami
2. Buat kesimpulan perihal apa yang anda dapatkan/pahami dari praktikum untuk materi **View** yang baru dilakukan, sertakan kesimpulan tersebut dalam rangkuman anda

BAB XII DATABASE SECURITY

12.1 Tujuan

Setelah praktikum ini usai, mahasiswa diharapkan:

- mampu memahami pentingnya keamanan basisdata
- mampu membuat *user* dan *role* di SQL
- mampu memahami perbedaan antara *user* dan *role* kaitannya dengan *permission*
- mampu memahami dan dapat menerapkan penggunaan perintah GRANT, REVOKE, dan DENY untuk mengatur hak akses di SQL

12.2 Dasar Teori

Database security (keamanan basisdata) adalah sekumpulan prosedur, aturan, *tools*, dan standar yang bersifat *established* (mapan) yang digunakan untuk melindungi data dari ancaman seperti pencurian data, penyalahgunaan, gangguan, aktivitas, dan serangan yang tidak diinginkan, dan lain sebagainya. *Database security* bersinggungan erat dengan *permission* dan *access* ke dalam struktur data dan data yang terkandung di dalamnya.

Ancaman terhadap basisdata dapat berupa ancaman eksternal (ancaman dari luar) dan internal (ancaman dari dalam). Berikut ini adalah beberapa ancaman yang mungkin terjadi pada basisdata:

- *hacker*
- *social engineers*
- *computer users (poor habit, password error (easy password), disregard for company policy, opening unknown email, inapropriate disclosure, procastination)*
- *network and database administrator*
- internet (halaman web (*webpage*), *misleading app, email, attachment, phising*)
- *malware (computer virus, worms, trojan, spyware, adware, bots)*

Seorang *database administrator* (DBA) harus mengerti pentingnya mengamankan basisdata. Mengamankan basisdata adalah bagian penting dari tugas seorang DBA, termasuk juga

memahami objek apa saja yang bisa dan harus diamankan, dan memahami pentingnya pengaturan ***account*** dan ***role*** dalam sebuah lingkungan basisdata.

Mengatur keamanan basisdata adalah tugas yang sulit sekaligus mudah. Mudah dalam hal menerapkan perintah keamanan (*security*), tetapi sulit dalam hal memastikan bahwa semua celah-celah keamanan yang rawan ditembus sudah tertutupi (*covered*) dengan sempurna.

Paparan di bawah ini akan membahas tentang *user managements*, mulai dari membuat *user* (*single and multiple*), menampilkan daftar *user* yang ada di basisdata, mengganti nama *user*, dan menghapus *user*.

CREATING USER

```
CREATE USER [IF NOT EXISTS] account_name IDENTIFIED BY 'password';
```

Contoh (1) membuat sebuah *user* baru

```
CREATE USER novi@localhost IDENTIFIED BY 'ini_passwd';
```

Contoh (2) membuat dua buah *user* baru

```
CREATE USER
       novi@localhost IDENTIFIED BY 'ini_passwd',
       prisma@localhost IDENTIFIED BY 'ini_passwd_juga';
```

Untuk membuat *multiple user* (dua buah *user* sekaligus atau lebih), caranya mirip dengan membuat *single user*, cukup pisahkan dengan koma antara satu *user* dengan *user* berikutnya.

SHOWING LIST OF USER

```
SELECT user, host FROM mysql.user;
```

Informasi tentang *user* akan tersimpan di basisdata mysql pada tabel user, tabel user memiliki 47 kolom dua di antaranya adalah kolom user dan host, kolom ini menyimpan nama-nama user dan host yang ada pada server basisdata.

RENAMING USER

```
RENAME USER old_user TO new_user;
```

Contoh:

```
RENAME USER novi@localhost TO nova@localhost;
```

DROPPING USER

```
DROP USER username;
```

SQL – Data Control Language (DCL)

Data Control Language (DCL) adalah komponen SQL yang mengatur akses terhadap basisdata. Perintah DCL antara lain adalah GRANT, REVOKE, dan DENY. GRANT adalah perintah yang digunakan untuk memberikan akses DDL maupun DML kepada *user* dan/atau *role* tertentu terhadap objek basisdata. REVOKE adalah perintah untuk menghapus/menghilangkan akses yang sebelumnya sudah diberikan kepada *user* atau *role* terhadap objek basisdata. Sementara DENY adalah perintah untuk melarang/mencegah diberikannya akses kepada *user* atau *role* tertentu.

GRANT

```
GRANT privilege [,privilege],..

ON privilege_level

TO account_name/role_name;
```

Privilege level menentukan objek basisdata di level apa yang dipilih. Ada 6 *privilege level* yang dapat dipilih saat memberikan akses yaitu: *global, database, table, column, store routine,* dan *proxy*.

Contoh (1) Memberi akses SELECT ke seluruh tabel di seluruh basisdata yang ada di server

```
GRANT SELECT
ON *.*
TO novi@localhost;
```

Contoh di atas adalah perintah GRANT dengan *privilege level* ***global***. *User* `novi@localhost` diberi akses SELECT ke seluruh basisdata yang ada di server. *Privildge level global* ditandai dengan *.* pada klausa ON

Contoh (2) Memberi akses INSERT ke database tertentu

```
GRANT INSERT
ON product_managements.*
TO novi@localhost;
```

Contoh di atas adalah perintah GRANT dengan *priviledge level* ***database***. Diberikan akses INSERT kepada `novi@localhost` terhadap basisdata `product_managements`. Akses INSERT ini berlaku untuk seluruh tabel yang ada di basisdata `product_managements`.

Contoh (3) Memberi akses UPDATE dan DELETE untuk basisdata dan tabel tertentu

```
GRANT UPDATE, DELETE
ON product_managements.customers
TO novi@localhost;
```

Contoh di atas adalah perintah GRANT dengan pri *priviledge level* ***table.*** Diberikan akses UPDATE, dan DELETE kepada `novi@localhost` terhadap basisdata `product_managements`, akses tersebut hanya diberikan untuk tabel tabel `customers`.

Contoh (4) Memberi akses SELECT untuk kolom tertentu pada sebuah tabel

```
GRANT SELECT (product_id,name,size,weight)
ON products
TO novi@localhost;
```

Contoh di atas adalah perintah GRANT dengan *privilege level column*. Diberikan akses SELECT kepada `novi@localhost` terhadap table products kolom `product_id`, `name`, `size`, dan `weight`. User `novi@localhost` hanya dapat mengakses tabel `products` dan kolom tertentu sebagaimana disebutkan pada perintah di atas. Untuk memberikan *privilege level column*, cukup tambahkan daftar kolom setelah klausa DML → `SELECT (product_id, name, size, weight)`. Jika kolom lebih dari satu maka cukup dipisahkan menggunakan koma. Sementara di bagian ON, cukup tuliskan nama tabelnya saja.

Contoh (5) Memberi akses untuk *function* dan/atau *stored procedure*

```
GRANT EXECUTE
ON PROCEDURE sp_insert_new_products
TO novi@localhost;
```

Perintah di atas adalah GRANT untuk *function/stored procedure*. Berdasarkan GRANT statement di atas, akan diberikan akses untuk mengeksekusi *stored procedure* dengan nama `sp_insert_new_products` kepada user `novi@localhost`.

SHOW GRANT for *spesific user*

```
SHOW GRANTS FOR novi@localhost;
```

REVOKE

```
REVOKE privilegee [,privilege]..
ON [object_type] privilege_level
```

```
FROM user1 [, user2] ..;
```

Contoh:

```
REVOKE INSERT, UPDATE
ON product_managements.*
FROM novi@localhost;
```

Perintah di atas akan menghapus/menghilangkan akses INSERT dan UPDATE yang dimiliki oleh *user* `novi@localhost` terhadap basisdata `product_managements`.

ROLE

Role adalah kumpulan hak akses. Sebagaimana *user account*, pemberian dan penghapusan hak akses dapat dilakukan terhadap *role*. *User account* dapat diberi hak akses terhadap *role*, satu buah *role* bisa terdiri dari beberapa *user account*, sehingga dengan memberikan hak akses kepada *role* bisa jadi merupakan aktivitas memberikan hak akses kepada beberapa *user account* sekaligus. *User account* biasanya merujuk pada user/individual tertentu, sedangkan *role* merujuk pada fungsi tertentu misalnya *finance*, *marketing*, *sales*, dan lain-lain.

CREATING ROLE

```
CREATE ROLE role_name,[role_name2}…;
```

Contoh:

```
CREATE ROLE finance_dept;
```

Perintah di atas adalah membuat *role* dengan nama `finance_dept`;

GRANTING ROLE TO USER

```
GRANT finance_dept TO user1@localhost, user2@localhost;
```

Perintah di atas berarti bahwa `user1@localhost` dan `user2@localhost` diberi akses role bernama `finance_dept`.

GRANTING PRIVILEGE TO ROLE

```
GRANT ALL
ON product_managements.salary
TO finance_dept;
```

Perintah di atas bearti bahwa role `finance_dept` diberi akses 'ALL' terhadap basisdata `product_managements` tabel `salary`. Jika dikaitkan dengan contoh sebelumnya tentang pemberian *role* kepada *user* `user1@localhost` dan `user2@localhost`, maka dua buah user tersebut juga memiliki akses 'ALL' terhadap basisdata dan tabel sebagaimana yang diberikan kepada role `finance_dept`.

12.3 Alat dan Bahan

Database tools	SQL atau MySQL
File import	product_managements.sql

12.4 Langkah Percobaan

Berikut ini adalah percobaan-percobaan yang akan kita lakukan:

1. Buat sebuah *user account* baru dengan nama `user1@localhost`, password: password
2. Dengan satu buah perintah saja, buat dua buah *user account* dengan nama `user2@localhost`, dan `user3@localhost` masing-masing dengan password: password
3. Tampilkan daftar *user account*
4. Berikan akses kepada `user1@localhost` agar bisa melakukan SELECT, UPDATE, dan DELETE di seluruh tabel pada basisdata `product_managements`

5. Berikan hak akses kepada `user2@localhost` agar bisa melakukan INSERT di tabel `products`
6. Berikan hak akses kepada `user3@localhost` agar bisa melakukan UPDATE di kolom `stock` tabel `products`
7. Tampilkan daftar akses dari masing-masing *user account* menggunakan perintah `SHOW GRANT`
8. Ganti nama `user3@localhost` menjadi `user4@localhost`
9. Hilangkan permission INSERT tabel `products` dari `user2@localhost`
10. Buktikan bahwa akses tersebut sudah hilang, gunakan perintah SHOW GRANT
11. Hapus `user3@localhost`, `user3@localhost`, dan `user3@localhost` dari tabel user
12. Buat sebuah *role* dengan nama `sales`
13. Dengan sebuah perintah CREATE, buat dua buah *user account* baru dengan nama `novi@localhost`, `prisma@localhost` password:password
14. Berikan akses role `sales` kepada `novi@localhost` dan `prisma@localhost`
15. Berikan akses untuk melakukan SELECT, UPDATE kepada *role* `sales` terhadap tabel `products` di basisdata `product_managements`
16. Menggunakan SHOW GRANT statement, buktikan bahwa poin 14 dan 15 berhasil.

12.5 Tugas

1. Buat rangkuman dari percobaan yang dilakukan, masing-masing lengkapi dengan narasi, *query*, dan *screenshoot* hasil eksekusi. Tuliskan rangkuman dengan lengkap, runut, dan mudah dipahami
2. Buat kesimpulan perihal apa yang anda dapatkan/pahami dari praktikum untuk materi **Database Security** yang baru dilakukan, sertakan kesimpulan tersebut dalam rangkuman anda

BAB XIII SQL BEST PRACTICE

13.1 Tujuan

Setelah praktikum ini usai, mahasiswa diharapkan:

- mampu memanfaatkan fitur/perintah pada SQL dengan optimal

13.2 Dasar Teori

Materi ini akan membahas beberapa *best practice* yang dapat diterapkan dalam SQL, sehingga dapat mencapai penggunaan SQL yang efektif, optimal dan teroptimasi dengan baik.

Penggunaan SELECT field daripada SELECT*

Penggunaan *asterisk* atau tanda bintang * biasanya nampak lebih sederhana dan mudah dalam penulisan SELECT statement, *user* tidak perlu menulis detail nama kolom sehingga *query* nampak lebih singkat, dan cepat untuk dieksekusi. Namun, penggunaan *asterisk* ini tidak disarankan dan dianggap *bad practice* khususnya pada *live production*. Pengaplikasian SELECT* akan mengembalikan seluruh kolom yang ada pada sebuah tabel, jika tabel memiliki banyak kolom dan baris, maka penggunaan *resource* yang sangat tidak efisien rentan terjadi karena menampilkan data yang tidak dibutuhkan

```
SELECT*FROM products;
```

SELECT statement di atas akan menampilkan seluruh data yang tersimpan di tabel products

```
SELECT product_id, name, weight, size, price, discount
FROM products;
```

SELECT statement di atas akan menampilkan data sesuai kebutuhan, yakni hanya dari kolom spesifik saja. Ketimbang SELECT*, SELECT statement di atas membutuhkan waktu eksekusi yang lebih sedikit.

Hindari SELECT DISTINCT

Klausa DISTINCT adalah klausa yang berfungsi untuk menampilkan nilai unik yang ada pada sebuah baris. Bagaimanapun, untuk mendapatkan nilai unik diperlukan sumber daya dan *processing power* yang besar. SELECT DISTINCT bekerja dengan cara mengelompokkan nilai yang sama dalam kelompok tertentu kemudian menampilkan hanya satu nilai di kelompok tersebut. Di sisi lain, pada beberapa kasus baris data ini sangat mungkin dikelompokkan ke dalam kelompok yang salah sehingga dapat mengakibatkan inakurasi data.

Berikut ini adalah contoh penggunaan SELECT DISTINCT yang tidak efisien:

```
SELECT DISTINCT full_name, province, district
FROM customers;
```

SELECT statement di atas akan mengembalikan baris sebagaimana SELECT statement tanpa DISTINCT. Untuk mendapatkan baris unik yang lebih akurat, sebaiknya SELECT statement tersebut diubah menjadi:

```
SELECT full_name, province, district, sub-district
FROM customers;
```

Dengan menghapus klausa DISTINCT dan menambahkan beberapa kolom, baris yang didapatkan otomatis adalah baris yang unik, tidak sama dengan baris lain di tabel tersebut. Menghapus klausa DISTINCT artinya juga meminimalisir penggunaan *resource* dan mempersingkatkan *processing phase* saat eksekusi *query*.

SELECT data *multiple table* menggunakan JOIN daripada WHERE

Di beberapa kasus, seorang SQL developer lebih menyukai penggunaan klausa WHERE ketimbang klausa JOIN untuk *retrieve data* dari *multiple table*. Misalnya:

```
SELECT customers.customer_id, customers.full_name, reviews.content
FROM customers, reviews
WHERE customers.customer_id = reviews.customer_id;
```

SELECT statement di atas akan menampilkan data dari kolom yang berasal dari tabel customers dan re*view*s yang memenuhi kondisi di klausa WHERE, yakni customer_id di dua tabel tersebut bernilai sama. SELECT statement di atas masuk dalam kategori *cartesian join* atau *cross join*. Pada cartesian join, semua kombinasi yang berpeluang dari dua tabel akan dibuat, kemudian filter diaplikasikan kepada baris kombinasi tersebut yakni hanya baris yang memenuhi kriteria WHERE yang akan ditampilkan.

Jika ada 200 baris data di tabel customers, dan 1000 baris data di tabel re*view*s, maka semua kemungkinannya akan ditampilkan, artinya 200 dikali 1000 sama dengan 200.000 baris. Kemudian filter diaplikasikan, dan hanya 200 baris data saja yang ditampilkan.

Cartesian join berpeluang menggunakan *resource* yang besar, dan akan sangat problematik jika diimplementasikan ke dalam tabel-tabel dengan *records* dalam jumlah yang besar. Alternatifnya adalah dengan menggunakan INNER JOIN, seperti di bawah ini:

```
SELECT customers.customer_id, customers.full_name, reviews.content
FROM customers
INNER JOIN reviews
ON customers.customer_id = reviews.customer_id;
```

SELECT statement di atas akan men-*generate* 200 baris data yang sesuai dengan klausa ON. Ini tentu jauh lebih cepat dan efisien, serta membutuhkan *resource* dan e*xecution time* yang jauh lebih sedikit ketimbang *cartesian join*.

Meskipun demikian, di beberapa sistem DBMS, sistem dapat mengenali klausa WHERE yang merupakan *cartesian join* dan otomatis diperlakukan seperti INNER JOIN. Namun di beberapa sistem DBMS join dengan klausa WHERE ini tidak dikenali sehingga penggunaan INNER JOIN lebih disarankan daripada join dengan klausa WHERE.

Penggunaan *wildchar character* di bagian akhir teks saja

Wildcard character adalah karakter yang digunakan untuk mengganti atau *subtitute* satu atau lebih string, *wildcard character* biasanya digunakan pada klausa LIKE yang memungkinkan opsi pencarian dengan *pattern* apapun dapat dilakukan. Meskipun wildcard character

memungkinkan untuk mendapatkan pencarian terluas, *wildchar character* juga dianggap sebagai pencarian yang paling tidak efisien. Untuk menggunakannya secara efisien, gunakan *wildchar* hanya di bagian akhir teks saja.

```
SELECT product_id, name, price, discount
FROM products
WHERE name LIKE "Avoskin%";
```

SELECT statement di atas akan mencari semua baris yang berawalan "Avoskin".

Penggunaan LIMIT untuk melihat contoh *result-set*

Biasanya yang dibutuhkan dalam sebuah SELECT statement hanya contoh hasil *query* dari sebuah tabel saja, bukan keseluruhan barisnya. Klausa LIMIT dapat digunakan untuk kasus ini, LIMIT dapat membatasi jumlah baris yang ingin ditampilkan tanpa harus menampilkan keseluruhan isi tabel, yang tentu saja akan membutuhkan sumber daya dan waktu yang besar. User dapat menentukan jumlah baris yang ingin ditampilkan dengan menambahkan angka setelah klausa LIMIT.

```
SELECT product_id, name, price, discount
FROM products LIMIT 3;
```

SELECT statement di atas akan mengembalikan hanya 3 baris teratas di tabel products, jika *sorting*-nya tidak didefinisikan maka urutannya sesuai dengan *primary key* di tabel products.

13.3 Alat dan Bahan

Database tools	MySQL
File import	product_managements.sql

13.4 Langkah Percobaan

Berikut ini adalah percobaan-percobaan yang akan:

kondisional

13.5 Tugas

3. Buat laporan praktikum dari percobaan yang dilakukan, masing-masing lengkapi dengan narasi, *query*, dan *screenshoot* hasil eksekusi. Tuliskan rangkuman dengan lengkap, runtut, dan mudah dipahami
4. Buat kesimpulan tentang a pa yang anda dapatkan/pahami dari praktikum untuk materi **SQL Best Practice** yang baru dilakukan, sertakan kesimpulan tersebut dalam rangkuman anda.

REFERENSI

https://www.red-gate.com/hub/product-learning/sql-prompt/finding-code-smells-using-sql-prompt-asterisk-select-list

https://docs.microsoft.com/id-id/sql/t-sql/language-elements/transactions-transact-sql?*view*=sql-server-ver16

https://docs.microsoft.com/id-id/windows/win32/cossdk/acid-properties

https://docs.microsoft.com/id-id/sql/t-sql/language-elements/variables-transact-sql?*view*=sql-server-ver16

https://docs.microsoft.com/id-id/sql/t-sql/language-elements/operators-transact-sql?*view*=sql-server-ver16

https://docs.microsoft.com/id-id/sql/t-sql/language-elements/if-else-transact-sql?*view*=sql-server-ver16

https://docs.microsoft.com/en-us/sql/t-sql/language-elements/cursors-transact-sql?redirectedfrom=MSDN&*view*=sql-server-ver16

Database security, Alfred Basta and Melissa Zgola, Cengage Learning.

https://www.sisense.com/blog/8-ways-fine-tune-sql-queries-production-databases/