

MODUL PRAKTIKUM

BASIS DATA

S1 REKAYASA PERANGKAT LUNAK



Published by school of computing



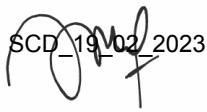
PENGESAHAN

Saya yang bertanda tangan di bawah ini:

Nama : Dr. Ati Suci Dian Martha, S.Kom., M.T.
NIP : 21830004
Koordinator Mata Kuliah : Basis Data
Prodi : S1 Rekayasa Perangkat Lunak

Menerangkan dengan sesungguhnya bahwa modul ini digunakan untuk pelaksanaan praktikum di Semester Genap Tahun Ajaran 2022/2023 di Laboratorium Informatika Fakultas Informatika Universitas Telkom.

Bandung, 3 Februari 2023

Mengesahkan Koordinator Mata kuliah Basis Data,  SCD_19/02/2023	Mengetahui Kaprodi S1 Rekayasa Perangkat Lunak,  Dr. Mira Kania Sabariah, S.T., M.T.
---	--

DAFTAR ISI

Lembar Pengesahan	
DAFTAR ISI	2
Modul 1. Aturan Praktikum	1
1.1 Aturan Praktikum	1
1.2 Visio	2
1.2.1 Membuat ER Diagram dengan Visio di dalam Microsoft Word	2
1.2.2 Membuat Skema Relational dengan Visio	6
1.3 Oracle	8
1.3.1 Pengenalan	8
1.3.2 Fungsionalitas	8
1.3.3 Kelebihan	8
1.3.4 Kekurangan	8
1.3.5 Instalasi Oracle	9
1.3.6 Cara menjalankan Oracle 11g	13
1.4 SQL Developer	14
1.4.1 SQL Developer pada praktikum ini digunakan untuk merancang logical model.	14
Modul 2. DDL/DML, Data types, Authorization	16
2.1. Ringkasan Materi	16
2.1.1. Pengenalan Oracle	16
2.1.1.1. Fungsionalitas	16
2.1.1.2. Kelebihan	16
2.1.1.3. Kekurangan	17
2.1.2. Tipe Data Pada Oracle 11g	17
2.1.3. Data Definition Language (DDL)	18
2.1.3.1. Create	18
2.1.3.1.1 Create Table	18
2.1.3.1.2 Create View	19
2.1.3.1.3 Create Sequence (Membuat Nomor Unik Secara Otomatis)	19
2.1.3.1.4 Create Synonym	20
2.1.3.1.5 Create User	20
2.1.3.1.6 Create Index	21
2.1.3.2. Drop	21
2.1.3.2.1 Drop Table	21

2.1.3.2.2 Drop View	21
2.1.3.2.3 Drop Sequence	21
2.1.3.2.4 Drop Synonym	21
2.1.3.2.5 Drop Index	22
2.1.3.2.6 Drop User	22
2.1.3.3. Alter	22
2.1.3.3.1 Alter Table	22
2.1.3.3.2 Rename	23
2.1.3.3.3 Truncate	24
2.1.3.3.4 Comment	24
2.1.3.3.5 Constraint	24
2.1.4 Data Manipulation Language (DML)	26
2.1.4.1. Pengenalan	26
2.1.4.2. Perintah-perintah DML	26
2.1.4.2.1 Insert	26
2.1.4.2.2 Update	26
2.1.4.2.3 Delete	26
2.1.4.2.4 Merge	27
2.1.5 View	27
2.1.5.1 Pengertian View	27
2.1.5.2 Jenis – Jenis View	27
2.1.5.2.1 Dynamic view	27
2.1.5.2.2 Materialized view	28
2.1.5.3 Create View	29
2.1.5.3.1 Dynamic view	29
2.1.5.3.2 Materialized view	29
2.1.6 Studi Kasus	29
2.1.7. Tutorial	31
Modul 3. Scripting SQL Basic Query – Pemrosesan Satu Tabel	35
3.1. Ringkasan Materi	35
3.1.1. SQL, SQL*Plus dan PL / SQL	35
3.1.2. Query Dasar	35
3.1.3. Contoh Query	36
3.1.4. Tentang Query	38
3.1.5. Klausa SELECT	38
3.1.6. Operator-Operator Dalam Oracle	39

3.1.7. Fungsi-Fungsi	42
3.2. Studi Kasus	46
Modul 4. Scripting SQL – Pemrosesan Banyak Tabel	47
4.1. Ringkasan Materi	47
4.1.1. Cross Join	47
4.1.2. Natural Join	48
4.1.3. Join Using	48
4.1.4. Join On	49
4.1.5. Outer Join	49
4.1.6. Left Outer Join	49
4.1.7. Right Outer Join	50
4.1.8. Full Outer Join	50
4.1.9. Non-Equijoin	50
4.2. Contoh Kasus:	51
4.3. Studi Kasus	52
4.4. Tutorial	53
Modul 5. Scripting Agregate Function	54
5.1. Ringkasan Materi	54
5.1.1. Jenis-jenis Group function	54
5.1.2. Syntax Group Function	55
5.1.3. Group Function dan tipe data	55
5.1.4. DISTINCT pada group function	55
5.1.5. Group function dan null values	55
5.1.6. GROUP BY rules	56
5.1.7. HAVING rules	58
5.1.8. Nesting Group Function	60
5.1.9. Contoh	60
5.2. Studi Kasus	61
5.3. Tutorial	62
Modul 6. Scripting Subquery & Nested Subquery	63
6.1. Ringkasan Materi	63
6.1.1. Masalah Subquery	63
6.1.2. Syntax Subquery	63
6.1.3. Basic Subquery rules	63
6.1.4. Subquery types: Single Row & Multiple Rows	63
6.1.5.1. Operator	64

6.1.5.2. GROUP BY pada Subquery	65
6.1.5.3. HAVING pada Subquery	66
6.1.6. Multiple Row subquery	66
6.1.6.1. IN operator	67
6.1.6.2. ANY operator	67
6.1.6.3. ALL operator	68
6.1.7. Multiple Column Subquery	69
6.1.8. <i>Subquery</i> pada DDL	70
6.1.9. <i>Subquery</i> pada DML	70
6.1.9.1. INSERT	70
6.1.9.2. UPDATE	70
6.1.9.3. DELETE	71
6.1.10. In-Line-View Subquery	71
6.1.11. Correlated Subquery	72
6.1.11.1. Syntax	72
6.1.11.2. Correlated Query	72
6.1.11.3. Correlated DML	73
1. Correlated UPDATE	73
2. Correlated DELETE	74
6.2. Contoh	74
6.3. Studi Kasus	78
6.4. Tutorial	78
Modul 7. Normalisasi	79
7.1. Ringkasan Materi	79
7.1.1. Tujuan Normalisasi	79
7.1.2. Tahapan dalam Normalisasi	79
7.2. Studi Kasus	83
7.3. Tutorial	85
Modul 8. Overview NoSQL	86
8.1. Ringkasan Materi	86
8.1.1. Jenis Basis Data NoSQL	86
Modul 9. NoSQL – Key Value Database	89
9.1. Ringkasan Materi	89
9.1.1. Apache Cassandra	89
9.1.2. Data Model	90
9.2. Tutorial	91

Modul 10. NoSQL – Columnar Database	93
10.1. Ringkasan Materi	93
10.1.1. Columnar Database vs Rows Database	93
10.2. Tutorial	94
Modul 11. NoSQL – Document Database	95
11.1. Ringkasan Materi	95
11.1.1 MongoDB	95
11.1.2 MongoDB Advantages	96
11.2 Tutorial 1	96
11.3 Tutorial 2	98
11.4 Tutorial 3	98
Modul 12. Scripting SQL (Create index, Create view, Analyze Table)	100
12.1. Ringkasan Materi	100
12.1.1. Analyze Table	100
12.1.2. Perintah Analyze	100
12.2. Studi Kasus	101
Modul 13. Scripting Procedure dan Function	102
13.1. Ringkasan Materi	102
13.1.1. Procedure	102
13.1.1.1 Parameter pada Procedure	103
13.1.1.2 Passing Parameter	104
13.1.1.3 Procedure dalam Procedure	105
13.2. Tutorial Procedure	105
13.3. Function	106
13.4. Studi Kasus	110
Daftar Pustaka	111

DAFTAR GAMBAR

Gambar 2-1. Contoh ER Diagram Sederhana	2
Gambar 2-2. Visio dari Word	3
Gambar 2-3. Chen type	3
Gambar 2-4. Shapes untuk membangun Erd	4
Gambar 2-5. Entitas	4
Gambar 2-6 Atribut	4
Gambar 2-7. Membuat Primary Key	5
Gambar 2-8 Shape lainnya	5
Gambar 2-10. Setelah disambungkan dengan relationship connector	6
Gambar 2-11. Ukuran Kertas	6
Gambar 2-12. Diagram akhir	7
Gambar 2-13. Contoh tabel relasi	7
Gambar 2-14. Pemilihan kategori diagram	8
Gambar 2-15. UML Database Notation	8
Gambar 2-16. Elemen utama	9
Gambar 2-17. Entitas	9
Gambar 2-18. ER Sederhana	10
Gambar 2-19. Oracle installation	11
Gambar 2-20. Configure security updates	12
Gambar 2-21. Oracle installation option	12
Gambar 2-22. Oracle installation system class	13
Gambar 2-23. Oracle typical installation	13
Gambar 2-24. Oracle installation summary	14
Gambar 2-25. Oracle installation	14
Gambar 2-26 Oracle installation	15
Gambar 2-27. SQL via CMD	15
Gambar 2-28. SQL Plus	16
Gambar 2-29. SQL dengan logged in user	16
Gambar 2-30. Data Model Browser	16
Gambar 2-31. Browser	17
Gambar 2-32. New Entity	17
Gambar 2-33. Elemen entitas	17
Gambar 2-34. Relasi	17
Gambar 2-35. Model sederhana	17
Gambar 3-1. Simple attributes, derived attributes, multivalue attributes dan composite attributes	19
Gambar 3-2. Unary	20
Gambar 3-3. Binary	20
Gambar 3-4. Ternary	21
Gambar 4-1. Notasi relasi 1 – ke – 1	24
Gambar 4-2. Penggambaran relasi 1 – ke – 1 pada Erd menggunakan notasi Chen	24
Gambar 4-3 Penggambaran relasi 1 – ke – 1 pada Erd menggunakan notasi Crows Feer	25
Gambar 4-4. Notasi relasi 1 – ke – n / n – ke – 1	25
Gambar 4-5. Penggambaran relasi n – ke – 1 pada Erd menggunakan notasi Chen	25
Gambar 4-6. Penggambaran relasi n – ke – 1 pada Erd menggunakan notasi Crows Feet	25
Gambar 4-7. Notasi relasi n – ke – n	25

Gambar 4-8 Penggambaran relasi n – ke – 1 pada ERD menggunakan notasi Chen	26
<i>Gambar 4-9. Penggambaran relasi n – ke – n pada ERD menggunakan notasi crows feet</i>	26
Gambar 4-10. Modalitas derajat 0 (1)	26
Gambar 4-11. Modalitas derajat 0 (2)	26
Gambar 4-12. Modalitas derajat 1 (1)	27
Gambar 4-13. Modalitas derajat 1 (2)	27
Gambar 4-14. Generalisasi account sebagai saving_account dan checking_account	27
Gambar 4-15. Spesialisasi pada account, saving_account dan checking_account	28
Gambar 5-1. Konversi atribut komposit	33
Gambar 5-2. Konversi atribut multivalues	34
Gambar 5-3. Relasi satu ke satu	34
Gambar 5-4. Hasil transformasi relasi satu ke satu dengan partisipasi total pada kedua pihak yang berelasi	34
Gambar 5-5. Hasil transformasi relasi 1 ke 1 dengan partisipasi total pada	35
Gambar 5-6. Hasil transformasi relasi 1 ke 1 dengan partisipasi sebagian pada kedua pihak yang berelasi	35
Gambar 5-7. Contoh ERD Relasi 1 ke 1[9]	35
Gambar 5-8. Contoh transformasi relasi 1 ke 1	36
Gambar 5-9. Relasi satu ke banyak	36
Gambar 5-10. Hasil Transformasi Relasi Satu ke Banyak	36
Gambar 5-11. Contoh ERD Relasi 1 ke n	36
Gambar 5-12. Contoh transformasi relasi satu ke banyak	37
Gambar 5-13. Relasi Banyak ke Banyak	37
Gambar 5-14. Hasil Transformasi Relasi Banyak ke Banyak	37
Gambar 5-15. Contoh ERD Relasi m ke n	38
Gambar 5-16. Contoh transformasi relasi banyak ke banyak untuk proses bisnis A	38
Gambar 5-17. Contoh transformasi relasi banyak ke banyak untuk proses bisnis B	38
Gambar 5-18. Contoh transformasi relasi banyak ke banyak untuk proses bisnis c	39
Gambar 5-19. Transformasi Relasi Unary dengan Kardinalitas satu ke banyak	39
Gambar 5-20. Transformasi Relasi Unary dengan Kardinalitas banyak ke banyak	39
Gambar 5-21. Contoh ERD dan hasil transformasi relasi unary dengan kardinalitas satu ke banyak	40
Gambar 5-22. Contoh ERD dan hasil transformasi relasi unary dengan kardinalitas banyak ke banyak	40
Gambar 5-23. ERD dan Transformasi relasi ternary	40
Gambar 5-24. Contoh transformasi entitas asosiatif	41
Gambar 5-25. Contoh transformasi entitas lemah	42
Gambar 5-26 Contoh kasus Gen-Spec	42
Gambar 5-27 Alternatif 1 Transformasi Gen-Spec	42
Gambar 5-28 Alternatif 2 Transformasi Gen-Spec	43
Gambar 6-1 Tahap normalisasi[8]	46
Gambar 6-2 Relasi Antar Tabel	48
Gambar 6-3 Relasi Antar Tabel Final	49
Gambar 9-1. Output query select	78
Gambar 9-2. Output query select dengan where	78
Gambar 9-3. Output query select dengan group by	79
Gambar 9-4. Output query select dengan order by	79
Gambar 9-5. Contoh implementasi	85
Gambar 9-6. Implementasi tanggal	86

Gambar 10-1. Implementasi case	90
Gambar 10-2. Implementasi decode	91
Gambar 10-3. Multiple Selection Condition	91
Gambar 11-1. Output	96
Gambar 10-2. Output left outer join	98
Gambar 10-3. Output right outer join	98
Gambar 11-4. Output full outer join	99
Gambar 11-5. Contoh kasus	100
Gambar 12-1. Hasil grup function	102
Gambar 12-2. Group by rules	104
Gambar 12-3. Nesting group function	107
Gambar 12-4. Skema	108
Gambar 13-1. Multiple column subquery	118
Gambar 13-2. Correlated subquery	120
Gambar 13-3. Syntax	121
Gambar 13-4. Correlated DML	122
Gambar 13-5. Correlated Delete	122
Gambar 13-6. Skema	123

Modul 1. Aturan Praktikum

Tujuan Praktikum

- Praktikan mampu memahami aturan praktikum

1.1 Aturan Praktikum

- Praktikum diampu oleh dosen kelas dan dibantu oleh asisten laboratorium dan asisten praktikum.
- Praktikum dilaksanakan pada MS.Teams dan Google Meet.
- Praktikan wajib membawa modul praktikum dan alat tulis.
- Praktikan diabsen oleh asisten praktikum.
- Durasi kegiatan praktikum S-1 = 1 SKS atau setara dengan 100 menit.
 - Terdapat Tugas Pendahuluan
 - 75 menit untuk penyampaian materi
 - 25 menit untuk pengerojan jurnal dan tes akhir
- Jumlah pertemuan praktikum:
 - 12 kali praktikum secara online (praktikum rutin)
 - 3 kali di luar lab (terkait Tugas Besar dan UAS)
 - 2 kali berupa presentasi Tugas Besar atau pelaksanaan UAS
- Praktikan wajib hadir minimal 75% dari seluruh pertemuan praktikum di lab. Jika total kehadiran kurang dari 75% maka nilai UAS/ Tugas Besar = 0.
- Praktikan yang datang terlambat :
 - <= 30 menit : diperbolehkan mengikuti praktikum tanpa tambahan waktu Tes Awal
 - > 30 menit : tidak diperbolehkan mengikuti praktikum
- Saat praktikum berlangsung, asisten praktikum dan praktikan:

9.1 Luring :

- Wajib menggunakan seragam sesuai aturan institusi.
- Wajib mematikan/ mengkondisikan semua alat komunikasi.
- Dilarang membuka aplikasi yang tidak berhubungan dengan praktikum yang berlangsung.
- Dilarang mengubah pengaturan *software* maupun *hardware* komputer tanpa ijin.
- Dilarang membawa makanan maupun minuman di ruang praktikum.**
- Dilarang memberikan jawaban ke praktikan lain.
- Dilarang menyebarkan soal praktikum.
- Dilarang membuang sampah di ruangan praktikum.
- Wajib meletakkan alas kaki dengan rapi pada tempat yang telah disediakan.

9.2 Daring :

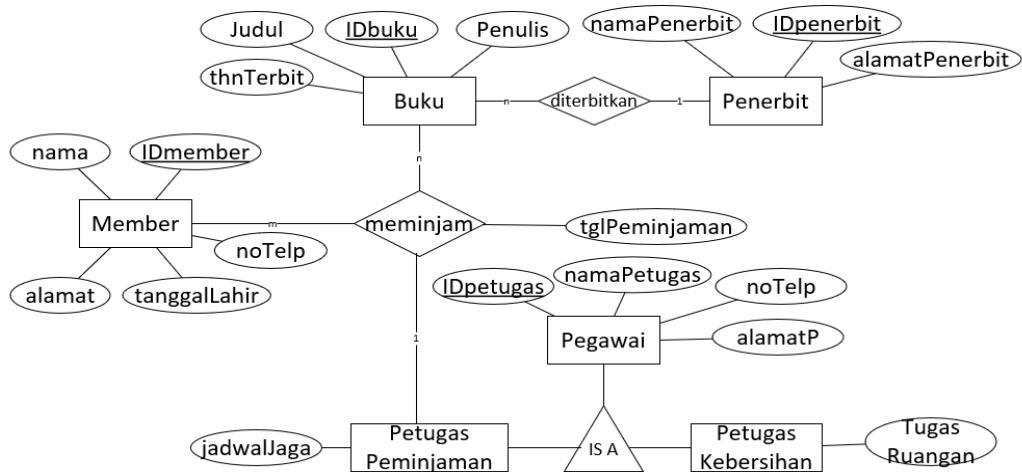
- Wajib menggunakan seragam sesuai aturan institusi.
 - Wajib menyalakan camera saat praktikum berlangsung.
 - Dilarang memberikan jawaban ke praktikan lain.
 - Dilarang menyebarkan soal praktikum.
- Setiap praktikan dapat mengikuti praktikum susulan maksimal dua modul untuk satu mata kuliah praktikum, dengan ketentuan:
 - Praktikan yang dapat mengikuti praktikum susulan hanyalah praktikan yang memenuhi syarat sesuai ketentuan institusi, yaitu: sakit (dibuktikan dengan surat keterangan medis), tugas dari institusi (dibuktikan dengan surat dinas atau dispensasi dari institusi), atau mendapat musibah atau kedukaan (menunjukkan surat keterangan dari orangtua/wali mahasiswa.)
 - Persyaratan untuk praktikum susulan diserahkan sesegera mungkin kepada asisten laboratorium untuk keperluan administrasi.
 - Praktikan yang diijinkan menjadi peserta praktikum susulan ditetapkan oleh Asman Lab dan Bengkel Informatika dan tidak dapat diganggu gugat.

Pelanggaran terhadap peraturan praktikum akan ditindak secara tegas secara berjenjang di lingkup Kelas, Laboratorium, Fakultas, hingga Universitas.

1.2 Visio

1.2.1 Membuat ER Diagram dengan Visio di dalam Microsoft Word

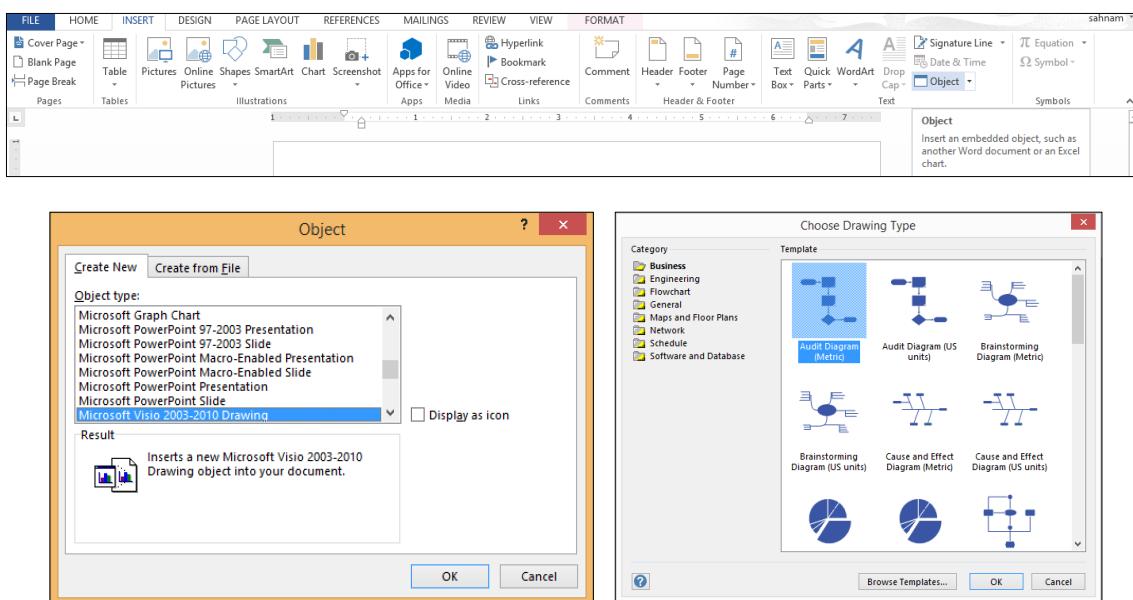
Kita akan membuat ER diagram seperti pada gambar di bawah ini menggunakan visio dan kemudian akan kita salin ke dalam bentuk PNG atau JPG. ER diagram pada Gambar 2-1 akan digunakan sebagai contoh pada subbab selanjutnya.



Gambar 1-1. Contoh ER Diagram Sederhana

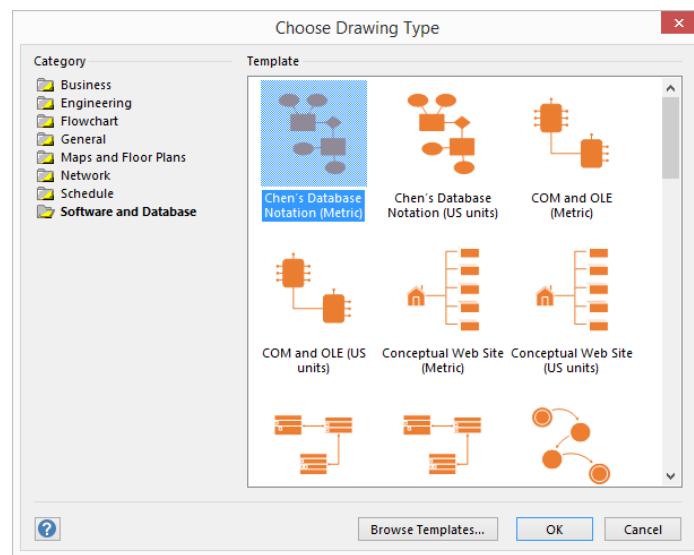
Adapun langkah-langkah yang dibutuhkan adalah sebagai berikut.

1. Buka Aplikasi Word anda dan pilih **Insert > Object > Microsoft Visio ... > Ok**. Maka akan muncul kotak dialog seperti di bawah ini



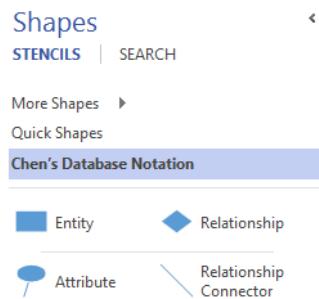
Gambar 1-2. Visio dari Word

2. Kemudian pilih **Software and Database** > **Chen's Database Notation** > Ok.



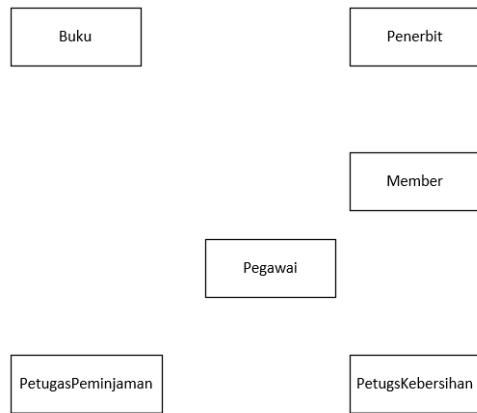
Gambar 1-3. Chen type

3. Maka akan muncul **Shapes** di sisi kiri aplikasi. Seperti yang dapat kita lihat terdapat 4 elemen utama dari shape **Chen's Database Notation**, yakni: *Entity*, *Relationship*, *Attribute*, dan *Relationship Connector*.



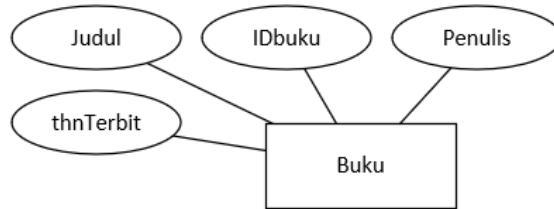
Gambar 1-4. Shapes untuk membuat ERD

4. **Drag Entity** untuk membuat entitas.



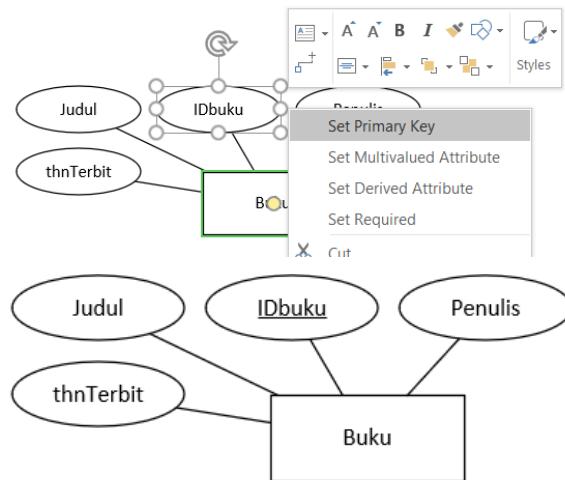
Gambar 1-5. Entitas

5. Drag **Attribute** untuk membuat atribut.



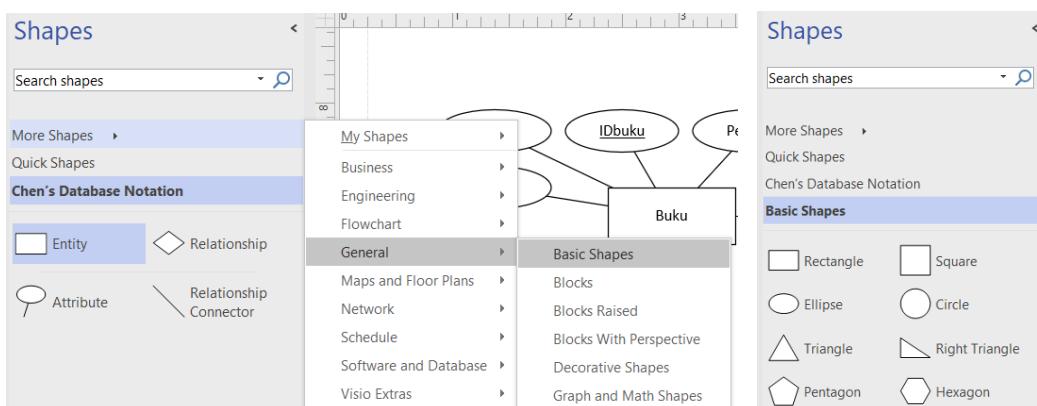
Gambar 1-6 Atribut

6. Terdapat beberapa jenis *attribute*, cara untuk melakukan *set attribute* adalah dengan klik kanan pada *shape* > lalu pilih jenis *attribute*.



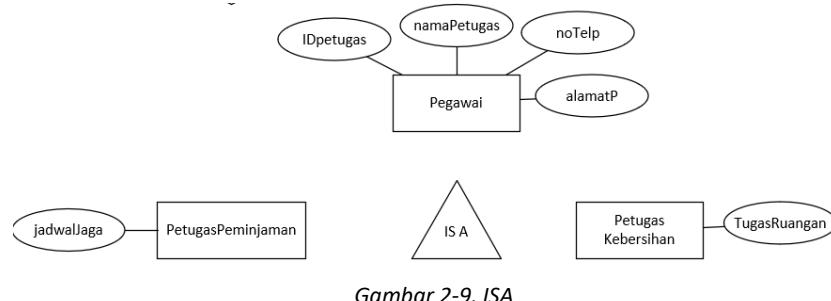
Gambar 1-7. Membuat Primary Key

7. Kemudian kita akan membuat **shape ISA**, namun tidak ada *shape* berbentuk segitiga pada *shape Chen's Database Notation*. Oleh karena itu, kita dapat menambahkan *shape* lain dengan cara klik **More Shapes > Pilih General > Basic Shapes** (pilih jenis *shape* sesuai kebutuhan).



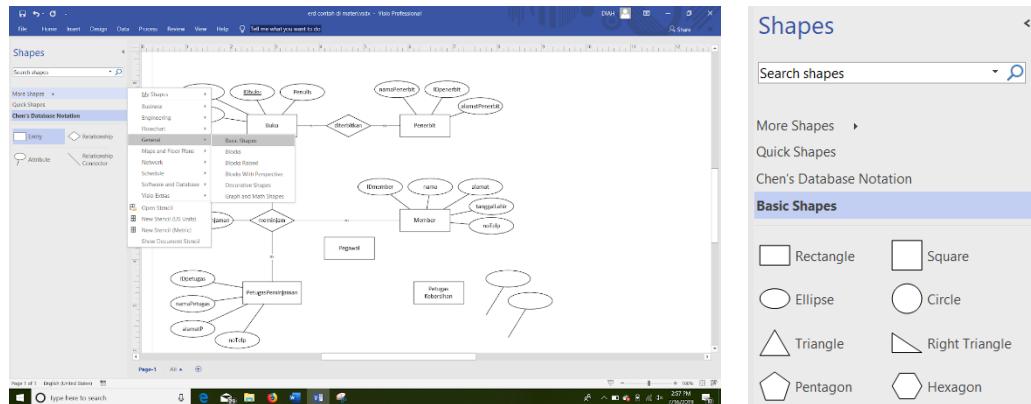
Gambar 1-8 Shape lainnya

8. **Drag Triangle** untuk membuat **shape ISA**.



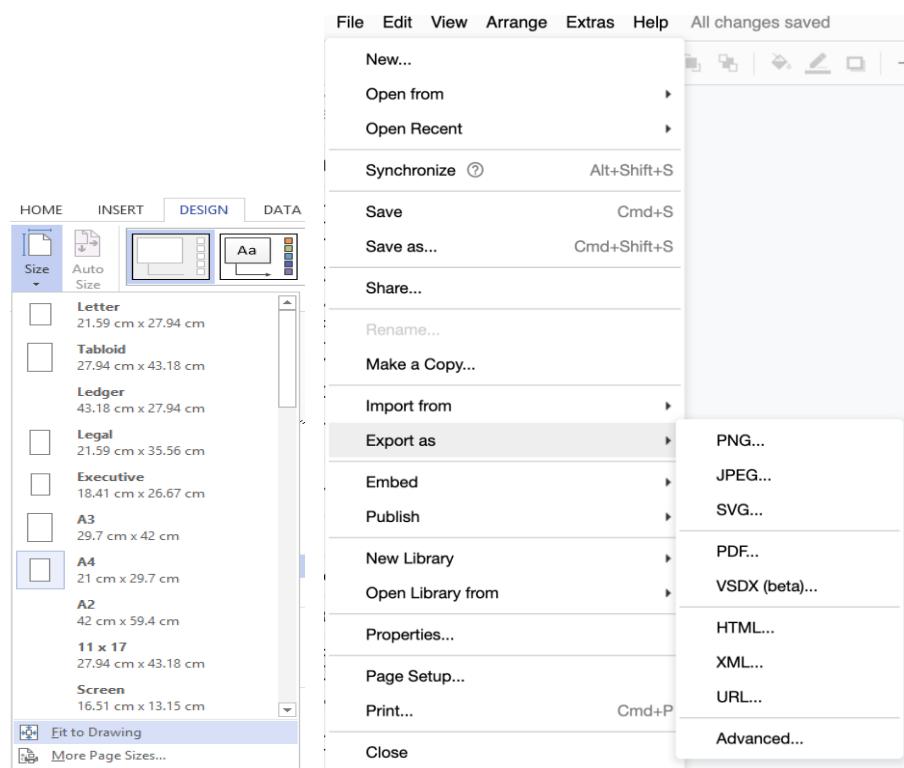
Gambar 2-9. ISA

9. Kemudian **Drag Relationship Connector** untuk membuat hubungan antar **shape**.



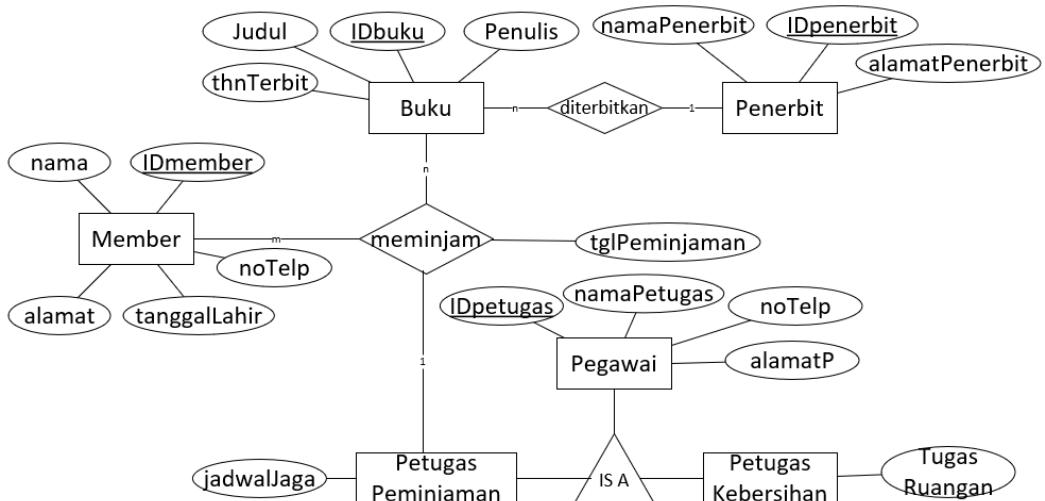
Gambar 1-9. Setelah disambungkan dengan relationship connector

10. Sesuaikan size kertas dengan diagram yang kita buat dengan cara klik **Design > Size > Fit to Drawing**. Kemudian **Save** pekerjaan anda dan **Close Visio**.



Gambar 1-10. Ukuran Kertas

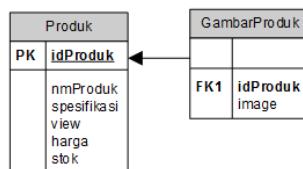
11. Berikut hasil akhir pembuatan diagram.



Gambar 1-11. Diagram akhir

1.2.2 Membuat Skema Relational dengan Visio

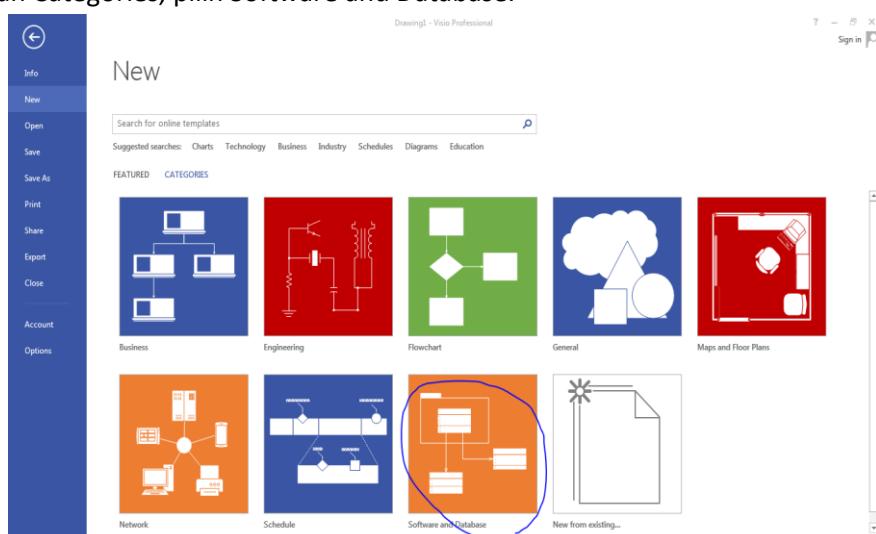
Kita akan coba membuat Skema Relasional seperti pada gambar di bawah ini dengan Visio



Gambar 1-12. Contoh tabel relasi

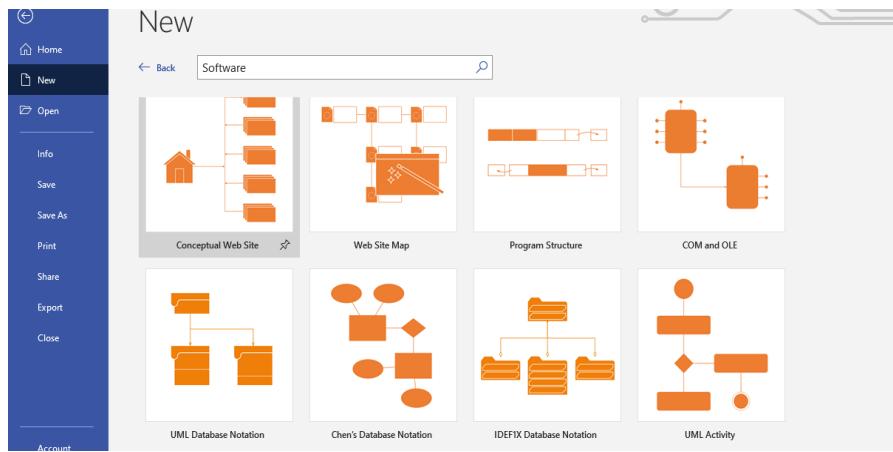
Adapun langkah-langkah yang dibutuhkan adalah sebagai berikut.

1. File -> New .
2. Pada bagian Categories, pilih Software and Database.



Gambar 1-13. Pemilihan kategori diagram

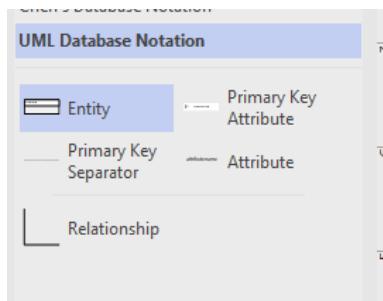
3. Lalu pilih **UML Database Notation**.



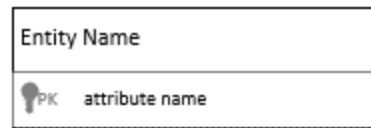
Gambar 1-14. UML Database Notation

Note: terdapat banyak jenis diagram yang dapat digunakan untuk menggambar ERD pada draw.io

4. Drag elemen yang diinginkan kedalam dokumen utama.



Gambar 1-15. Elemen utama

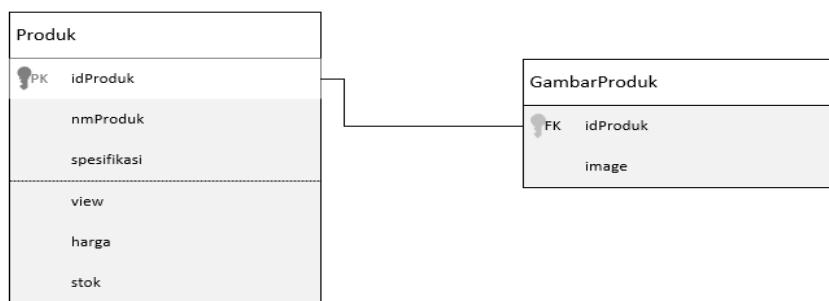


Gambar 1-16. Entitas

5. Modifikasi Nama Entitas dan atribut.

6. Tambahkan elemen lain seperti entitas lain atau relasi

7. Hasil akhir



Gambar 1-17. ER Sederhana

1.3 Oracle

1.3.1 Pengenalan

Oracle *Database* adalah suatu perangkat lunak *database* yang digunakan untuk menyimpan data. Biasanya perusahaan besar yang menggunakan program ini. Aplikasi ini merupakan produk yang dikembangkan oleh Oracle Corporation, pertama kali dirilis pada tahun 1979. Oracle dikembangkan oleh Larry-Ellison bersama Bob Miner, dan Ed Oates yang pada awalnya bekerja di *Software Development Laboratories* (SDL) pada tahun 1977. SDL kemudian berhasil membuat versi *prototype* orisinil dari Oracle Software, yaitu Oracle versi 1, yang ditulis dalam bahasa C dan mempunyai SQL *interface*. Nama Oracle berasal dari *code-name* suatu proyek milik CIA dimana Larry-Ellison dulunya bekerja.

1.3.2 Fungsionalitas

Oracle *Relational Database Management System* (Oracle RDBMS) atau sederhananya Oracle, merupakan sebuah RDBMS yang mempunyai fungsi antara lain sebagai berikut :

1. *Data Storage* pada area perancangan basis data.
2. *Access Data* untuk aplikasi tertentu, dengan Teknik Optimisasi.
3. *Database Security* dan pengaturan *User Privilege* tertentu.
4. *Consistency Data* yang berfungsi untuk proteksi data, dan *locking-mechanism*.
5. *Integrity Data* yang berfungsi pada implementasi basis data tersebut.

1.3.3 Kelebihan

1. *Scalability*, kemampuan menangani banyak *user* yang melakukan koneksi secara simultan tanpa berkurangnya *performance* secara signifikan. Dalam dokumentasinya, Oracle menyebutkan bahwa database Oracle dapat melayani puluhan ribu *user* secara simultan.
2. *Reliability* yang bagus, yaitu kemampuan untuk melindungi data dari kerusakan jika terjadi kegagalan fungsi pada sistem seperti *disk failure*.
3. *Stability*, yaitu kemampuan untuk tidak *crash* karena beban yang tinggi.
4. *Availability*, yaitu kemampuan dalam penanganan *crash* atau *failure* agar *service* tetap berjalan normal.
5. *Multiplatform*, dapat digunakan pada banyak sistem operasi seperti Windows, Unix, Linux dan Solaris.
6. Mendukung data yang berukuran besar. Berdasarkan dokumentasinya, Oracle dapat menampung data sampai 512 petabyte (1 petabyte= 1024 terabyte).
7. *Security* yang cukup handal.

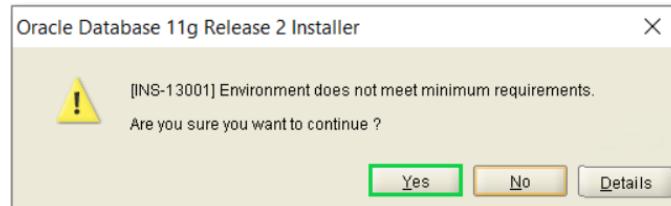
1.3.4 Kekurangan

1. Merupakan *software* DMBS yang paling mahal, paling rumit, dan paling sulit untuk dipelajari.
2. Membutuhkan spesifikasi *hardware* yang tinggi untuk dapat menjalankan *software* DMBS Oracle supaya berjalan dengan stabil.
3. Hanya diperuntukan bagi perusahaan berukuran besar, dan tidak cocok untuk perusahaan kecil maupun menengah.
4. Data yang bertambah ukurannya akan mengalami kelambatan proses, jadi harus ada *database management*.

5. Harga yang sangat mahal untuk sebuah *database* dan penggunaan Oracle sangat memakan banyak biaya, mulai dari *device* sampai diperlukannya DBA yang handal.

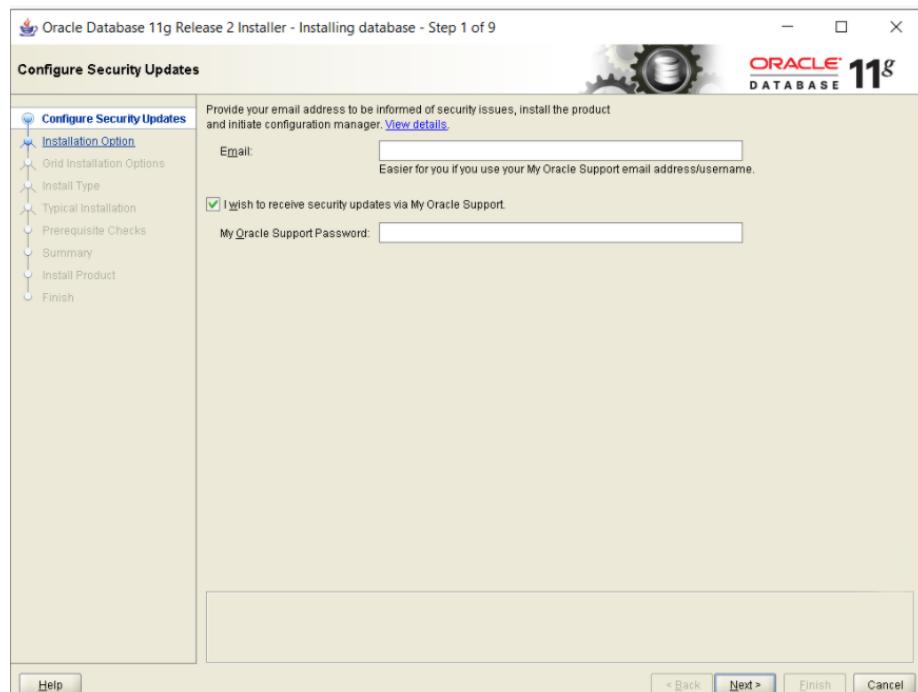
1.3.5 Instalasi Oracle

1. Double click pada setup oracle yang telah diunduh setup.exe
2. Setelah itu click yes pada oracle installation



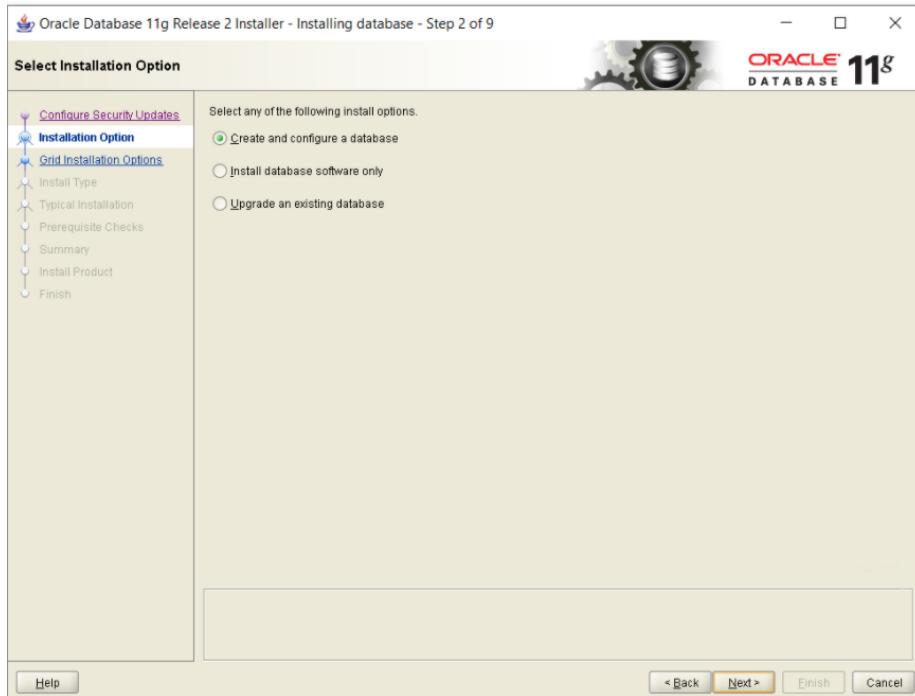
Gambar 1-18. Oracle installation

3. Masukkan email untuk mendapatkan update dari oracle (opsional)



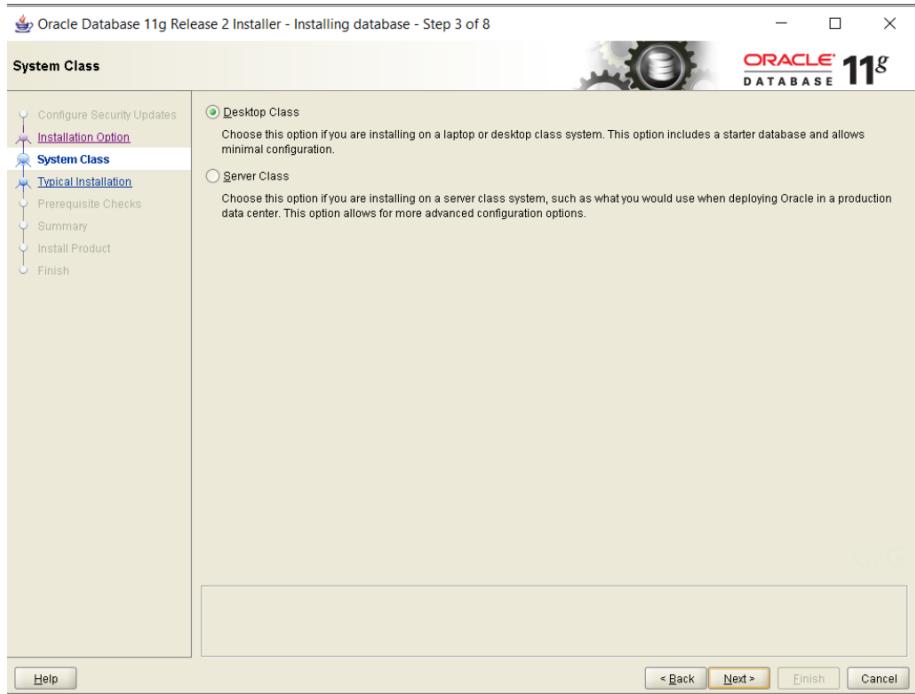
Gambar 1-19. Configure security updates

4. Create and configure a database



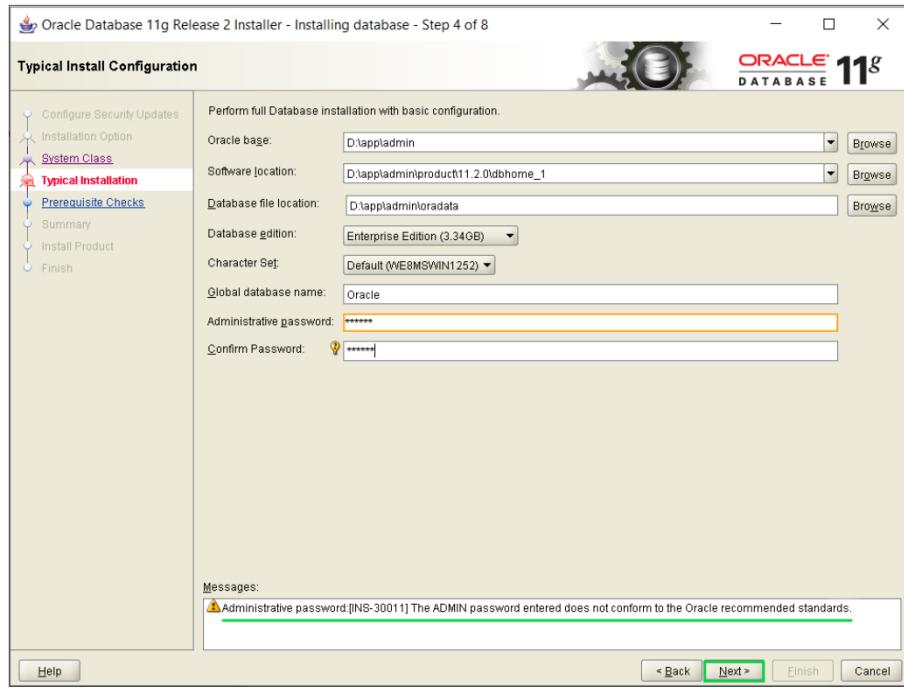
Gambar 1-20. Oracle installation option

5. Pilih dekstop class



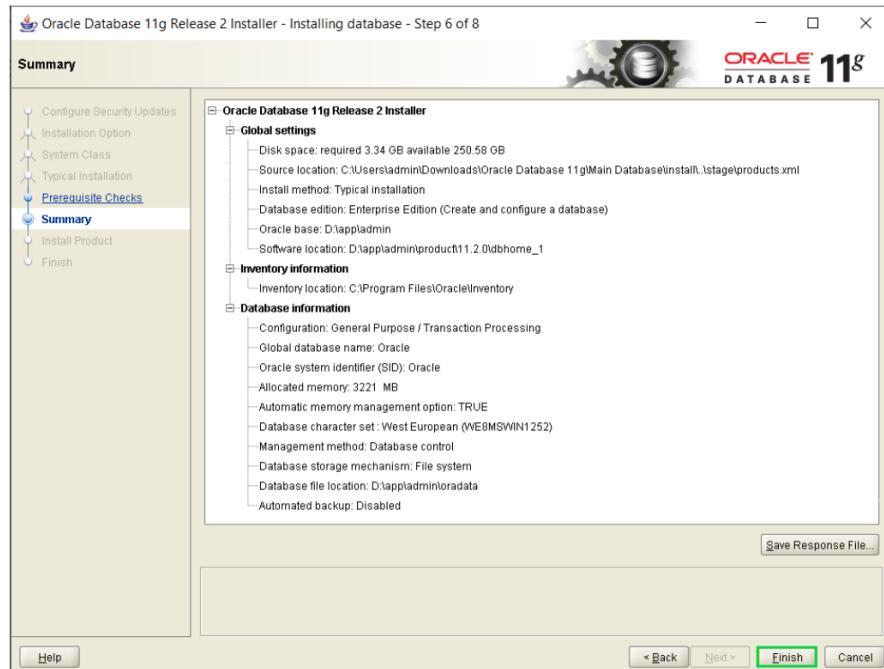
Gambar 1-21. Oracle installation system class

6. Konfigurasi sesuai kebutuhan user



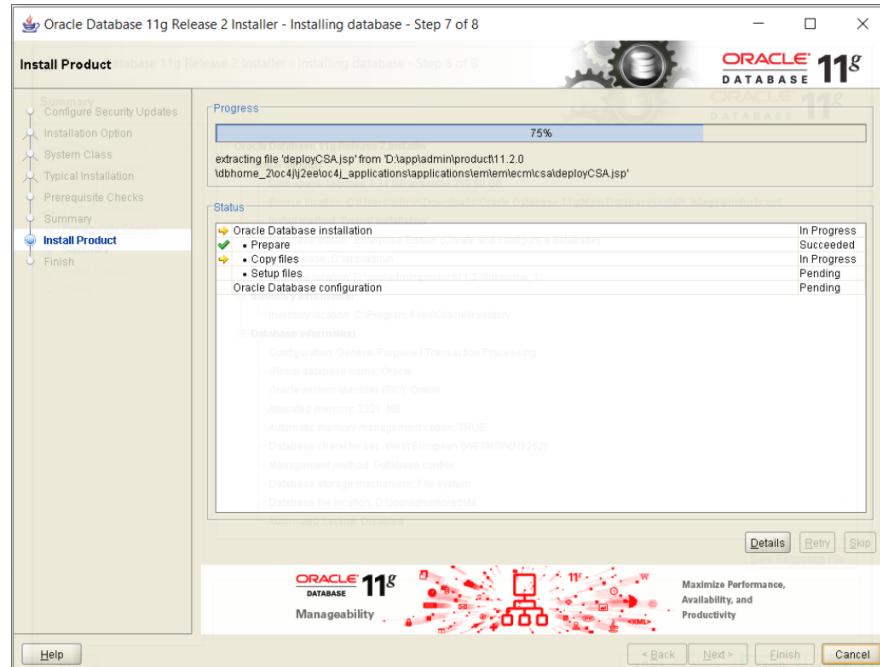
Gambar 1-22. Oracle typical installation

7. Pengecekan hardware oleh oracle



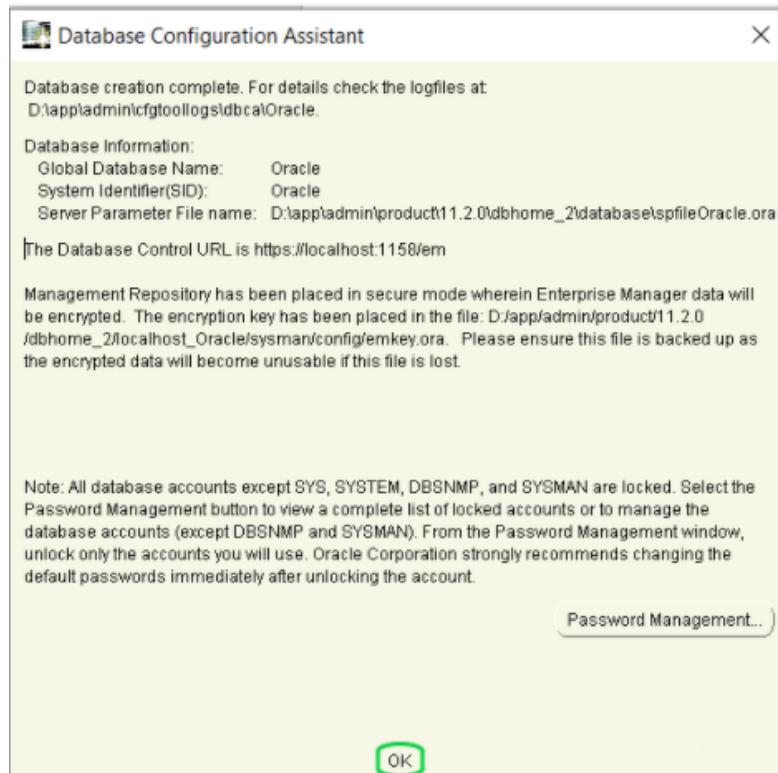
Gambar 1-23. Oracle installation summary

8. Oracle installation



Gambar 1-24. Oracle installation

9. Click ok untuk menyelesaikan installasi



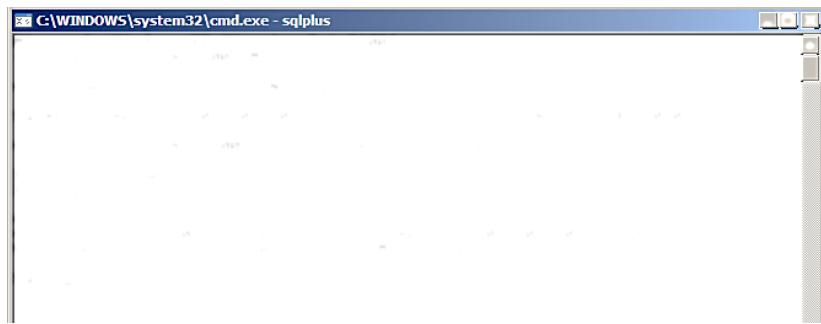
Gambar 1-25 Oracle installation

1.3.6 Cara menjalankan Oracle 11g

Ada 3 cara menjalankan Oracle pada PC,yaitu :

1. *Command prompt*

Untuk membuka *command prompt*, klik **start->Run** ketik **cmd**. Setelah muncul, ketik: **SQLPLUS**, masukkan **username** (*default*-nya adalah **system**), dan **password** (jika diminta) yang dibuat saat membuat *database*, setelah itu masuk ke *database* Oracle ditandai dengan “**SQL>**”. Perintah SQL bisa dijalankan setelah tulisan “**SQL>**” muncul seperti pada gambar 2-1.



Gambar 1-26. *SQL via CMD*

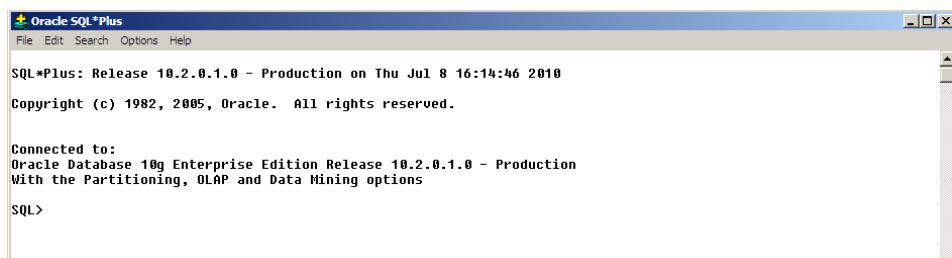
2. **SQL*PLUS**

Menggunakan *SQL*Plus* hampir sama saja seperti *Command prompt* dimana sintaks SQL yang akan di *running* harus diketikkan manual, hanya saja *SQL*Plus* ini *tools* bawaan dari Oracle. Untuk mengaksesnya pilih **Start □ All Program □ Oracle - OraDb10g_home1 □ Application Development □ SQL Plus**.



Gambar 1-27. *SQL Plus*

Pada *SQL*plus* ini harus dimasukkan *username* dan *password* yang telah dibuat pada saat instalasi Oracle (atau saat membuat *database*). Untuk penggunaan PC *standalone* (yang tidak terhubung jaringan) tidak perlu memasukkan *host string*.



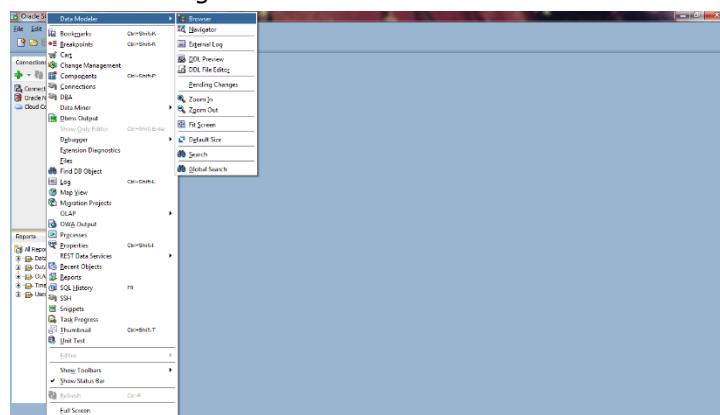
Gambar 1-28. *SQL dengan logged in user*

1.4 SQL Developer

1.4.1 SQL Developer pada praktikum ini digunakan untuk merancang logical model.

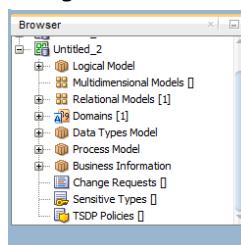
Berikut merupakan dasar penggunaan SQL Developer.

1. Aktifkan *Data Modeler view* dengan cara: Klik *View -> Data Modeler -> Browser*



Gambar 1-29. Data Model Browser

2. Akan muncul menu bar baru.
3. Klik kanan pada menu *Design -> New Design*



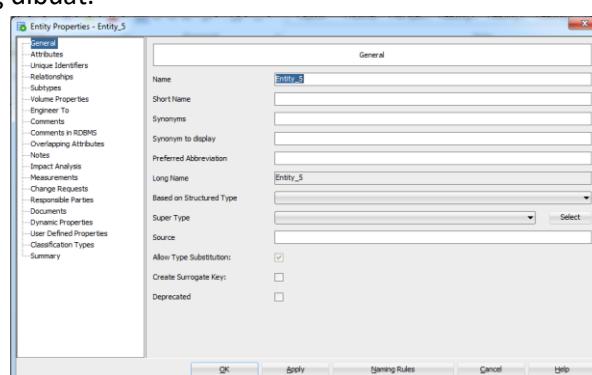
Gambar 1-30. Browser

4. Pada bagian Logical Model, klik kanan -> Show
5. Nantinya akan muncul field baru di field utama, dan muncul toolbox baru dibagian atas.
6. Untuk membuat entitas baru, klik *New Entity*, dan klik ke field kosong.



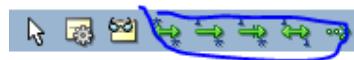
Gambar 1-31. New Entity

7. Akan muncul windows baru yang dapat mengatur entitas mulai dari nama, hingga atribut yang ada pada entitas yang dibuat.



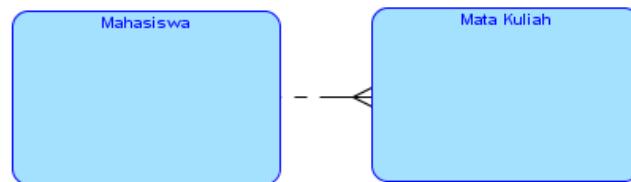
Gambar 1-32. Elemen entitas

8. Untuk membuat relasi, pilih salah satu dari relasi yang ada pada *toolbox*.



Gambar 1-33. Relasi

9. Hasil akhir sederhana.



Gambar 1-34. Model sederhana

Modul 2. DDL/DML, Data types, Authorization

Tujuan Praktikum

1. Mampu menjelaskan konsep dasar Oracle 11g
2. Mampu mengoperasikan DDL dalam Oracle dan mengaplikasikannya di studi kasus
3. Mengetahui dan memahami operasi-operasi *Data Manipulation Language* (DML)

2.1. Ringkasan Materi

2.1.1. Pengenalan Oracle

Oracle *Database* adalah suatu perangkat lunak *database* yang digunakan untuk menyimpan data. Biasanya perusahaan besar yang menggunakan program ini. Aplikasi ini merupakan produk yang dikembangkan oleh Oracle Corporation, pertama kali dirilis pada tahun 1979. Oracle dikembangkan oleh Larry-Ellison bersama Bob Miner, dan Ed Oates yang pada awalnya bekerja di Software Development Laboratories (SDL) pada tahun 1977. SDL kemudian berhasil membuat versi *prototype* orisinil dari Oracle software, yaitu Oracle versi 1, yang ditulis dalam bahasa C dan mempunyai SQL *interface*. Nama Oracle berasal dari *code-name* suatu proyek milik CIA dimana Larry-Ellison dulunya bekerja.

2.1.1.1. Fungsionalitas

Oracle Relational Database Management System (Oracle RDBMS) atau sederhananya Oracle', merupakan sebuah RDBMS yang mempunyai fungsi antara lain sebagai berikut:

1. *Data Storage* pada area perancangan basis data.
2. *Access Data* untuk aplikasi tertentu, dengan Teknik Optimisasi.
3. *Database Security* dan pengaturan *User Privilege* tertentu.
4. *Consistency Data* yang berfungsi untuk proteksi data, dan *locking-mechanism*.
5. *Integrity Data* yang berfungsi pada implementasi basis data tersebut.

2.1.1.2. Kelebihan

1. *Scalability*, kemampuan menangani banyak *user* yang melakukan koneksi secara simultan tanpa berkurangnya *performance* secara signifikan. Dalam dokumentasinya, Oracle menyebutkan bahwa *database* Oracle dapat melayani puluhan ribu *user* secara simultan.
2. *Reliability* yang bagus, yaitu kemampuan untuk melindungi data dari kerusakan jika terjadi kegagalan fungsi pada sistem seperti *disk failure*.
3. *Stability*, yaitu kemampuan untuk tidak *crash* karena beban yang tinggi.
4. *Availability*, yaitu kemampuan dalam penanganan *crash* atau *failure* agar *service* tetap berjalan normal.
5. *Multiplatform*, dapat digunakan pada banyak sistem operasi seperti Windows, Unix, Linux dan Solaris.
6. Mendukung data yang berukuran besar. Berdasarkan dokumentasinya, Oracle dapat menampung data sampai 512 petabyte(1 petabyte= 1024 terabyte).
7. *Security* yang cukup handal.

2.1.1.3. Kekurangan

1. Merupakan *software* DMBS yang paling mahal, paling rumit, dan paling sulit untuk dipelajari.
2. Membutuhkan spesifikasi *hardware* yang tinggi untuk dapat menjalankan *software* DMBS Oracle supaya berjalan dengan stabil.
3. Hanya diperuntukan bagi perusahaan berukuran besar, dan tidak cocok untuk perusahaan kecil maupun menengah.
4. Data yang bertambah ukurannya akan mengalami kelambatan proses, jadi harus ada *database management*.
5. Harga yang sangat mahal untuk sebuah *database* dan penggunaan Oracle sangat memakan banyak biaya, mulai dari *device* sampai diperlukannya DBA yang handal.

2.1.2. Tipe Data Pada Oracle 11g

1. *Char (i)*

String dengan panjang karakter **tetap** sebesar ukuran *i*. Tipe data ini mempunyai kemampuan yang hampir sama yakni menyimpan karakter, hanya saja maksimum *size* yang disimpan hanya sebesar **2000 bytes**.

2. *Varchar2 (i)*

String yang memiliki panjang karakter **variable** dengan panjang **maximal** sebesar *i*. Tipe data ini memperbolehkan penyimpanan semua karakter yang dapat dimasukkan melalui keyboard. Maksimum *size* yang dapat disimpan sebesar **4000 bytes(karakter)**. Tipe data ini juga memungkinkan untuk menyimpan data numerik. Biasanya *space* akan langsung dielimansi jika menggunakan tipe data ini.

Contoh : "Informaticslab " (dengan 3 spasi di akhir) dimasukkan ke dalam kolom yang didefinisikan dengan tipe data *varchar2*, maka akan menjadi "Informaticslab" (tidak ada spasi).

3. *Number(p,s)*

tipe data yang memiliki presisi *p* dan *s* digit dibelakang koma. Jika kita abaikan *p* dan *s* berarti dianggap sebagai tipe *number floating point*. Tipe data ini menyimpan bilangan integer sampai maksimum dari digit integer tersebut. Bagian kiri menunjukkan presisi dan bagian kanan menunjukkan skala.

Contoh:

Number specification	Column Length	Decimal Digits
(4,2)	4	2
(7,4)	7	4
(15,11)	15	11

Jumlah integer yang disajikan adalah merupakan **selisih** antara kedua angka tersebut. Misalnya untuk spesifikasi **(7,4)** artinya **3 digit sebelum koma** dan **4 digit dibelakang koma**. Jika jumlah digit di belakang koma melebihi daripada yang sudah didefinisikan maka akan langsung dibulatkan.

4. *Long*

Data karakter dengan ukuran panjang bervariasi, hingga mencapai **2 GigaBytes**. (tipe data ini tidak dapat digunakan sebagai *key*).

5. *Raw(size)*

Tipe data berupa *binary* data dengan ukuran maksimal **255 bytes**.

6. Long Raw

Tipe data berupa *binary* data dengan ukuran maksimal **2 GigaBytes** tidak dikonversi oleh oracle (data mentah apa adanya, spasi dihitung 1 karakter).

7. Date

Tipe data ini meyimpan **waktu** dan **tanggal** dari sebuah informasi, dengan komponen waktu yang dibulatkan ke detik terdekat. Untuk menampilkannya menjadi sebuah *text*, harus menggunakan fungsi **to_char**.

8. Timestamp

Tipe data ini mirip dengan tipe data **date**. Di dalam tipe data ini terdapat komponen waktu yang dapat langsung ditampilkan tanpa harus mengubahnya terlebih dahulu dengan fungsi **to_char**.

9. Clob

Tipe data ini memperbolehkan penyimpanan objek dengan ukuran yang sangat besar sekitar **4 GB**. **Clob** mirip sekali dengan varchar2 hanya saja tipe data ini mempunyai kapasitas maksimum yang lebih besar.

10. Blob

Tipe data ini memperbolehkan penyimpanan objek *binary* yang besar dan tidak terstruktur. Contoh dari tipe data **blob** adalah **suara** dan **video**.

2.1.3. Data Definition Language (DDL)

DDL merupakan bagian dari perintah SQL untuk membuat, memodifikasi atau menghapus struktur basis data Oracle. Suatu DDL akan dieksekusi apabila sudah diakhiri dengan ';' (titik koma).

2.1.3.1. Create

2.1.3.1.1 Create Table

Perintah Create Table digunakan untuk **membuat/mendefinisikan sebuah tabel baru**.

Sintaks :

```
|-----|  
| CREATE TABLE nama_tabel  
| (Nama_atribut1 tipe_data CONSTRAINT_pk_nama_tabel PRIMARY KEY,  
| Nama_atribut2 tipe_data [unique] [notnull/null] [default] <nilai default>  
| [check],  
| .....  
| );  
|-----|
```

Contoh :

```
CREATE TABLE SUPERVISOR  
(  
    NIP CHAR(6) NOT NULL ENABLE,  
    NMSUPERVISOR VARCHAR2(20),  
    THN_MASUK NUMBER,  
    CONSTRAINT SUPERVISOR_PK PRIMARY KEY (NIP) ENABLE  
)
```

Ket: Fungsi dari **CONSTRAINT** akan di bahas pada sub-bab V.

NOTE: Untuk melihat struktur sebuah *table* yang sudah dibuat dapat digunakan perintah **DESCRIBE nama_tabel;** atau **DESC nome_tabel;**

2.1.3.1.2 Create View

View merupakan sebuah **tabel semu / tabel logik**, dimana datanya berasal dari satu atau lebih tabel lain sebagai tabel sumber. View tidak memiliki indeks, terdapat batasan dalam memperbarui data dalam view. Ketika view diperbarui maka perubahannya dialurkan ke tabel dasar yang mendasari referensi untuk membuat view. Tabel dasar merupakan tabel dalam model data relasional yang berisikan *raw data*.

Sintaks :

```
| CREATE [OR REPLACE] [FORCE] [NOFORCE] VIEW nama_view  
| [(alias[,alias]...)] AS  
| Subquery [with <CHECK option>|<READ ONLY>];
```

Contoh:

```
CREATE OR REPLACE FORCE VIEW view_data_transaksi  
    (Id_transaksi, nama_member, tgl_trans, diskon)  
AS  
    select T.idtransaksi, M.nmmember, T.tgltransaksi, S.diskon  
    from transaksi T, member M, membership S  
    where T.idmember=M.idmember  
    and S.idmembership=M.idmembership  
    with check option;
```

2.1.3.1.3 Create Sequence (Membuat Nomor Unik Secara Otomatis)

Sequence digunakan untuk **men-generate angka angka unik secara otomatis**. Biasanya sequence diaplikasikan untuk membentuk *primary key* secara otomatis.

Sintaks :

```
| CREATE SEQUENCE nama_sequence  
| [increment by integer]  
| [start with integer]  
| [maxvalue integer | nomaxvalue]  
| [minvalue integer | nominvalue]  
| [cycle | nocycle]  
| [cache integer | nocache]  
| [order | noorder]  
| ;
```

Contoh :

```
CREATE SEQUENCE idMembership  
    Increment by 1  
    Start with 1  
    Maxvalue 999999  
    Minvalue 0 Nocycle  
    Nocache  
    Order;
```

Pada sintak pembentukan sequence terdapat beberapa kata kunci seperti berikut:

- **INCREMENT BY**, untuk menyatakan nilai penambahan/penurunan pada sequence.
- **START WITH**, untuk menyatakan nilai awal yang dihasilkan sequence.
- **MAXVALUE**, untuk menyatakan nilai tertinggi sequence. **NOMAXVALUE** berarti nilai

tertinggi yang dihasilkan tidak dibatasi. *MAXVALUE* dan *NOMAXVALUE* bersifat optional.

- **MINVALUE**, untuk menyatakan nilai terendah yang dihasilkan *sequence*. *MINVALUE* berarti nilai terendah yang dihasilkan tidak dibatasi. *MINVALUE* dan *NOMINVALUE* bersifat optional.
- **CYCLE**, untuk menyatakan sequence akan kembali ke nilai terendah yang ditetapkan. **NOCYCLE** adalah kebalikan dari **CYCLE**.
- **CACHE**, untuk menentukan berapa banyak nilai dari sequence yang akan dialokasikan ke memori. **NOCACHE** berarti tidak ada nilai sequence yang dialokasikan ke memori.
- **ORDER** atau **NOORDER** berfungsi untuk menjamin atau tidak, nilai yang *di-generate* adalah berurut berdasarkan permintaan pada *server paralel*. **Default** dari klausa ini adalah **NOORDER**.

Untuk penggunaan sequence, dapat menggunakan :

```
| Sintaks : Nama_sequence.nextval
```

2.1.3.1.4 Create Synonym

Synonym digunakan untuk membuat nama alias bagi sebuah tabel, view atau sequence.

Synonym biasanya digunakan untuk menjaga keamanan struktur suatu tabel.

Sintaks :

```
| CREATE [PUBLIC] SYNONYM nama_synonim  
| For nama_tabel/view/sequence;
```

Contoh :

```
CREATE SYNONYM prdk FOR produk;
```

Tujuan penggunaan synonym adalah:

- Untuk mengganti atau menyingkat nama yang sulit diingat dari sebuah objek.
- User dapat mengakses data pada *table* yang berisi informasi yang sensitive dan private melalui synonym tanpa harus mengetahui mana table aslinya sehingga kamu tidak dapat melakukan perubahan pada struktur *table*.

2.1.3.1.5 Create User

Create user digunakan untuk mendefinisikan user baru beserta password-nya.

Sintaks :

```
| CREATE USER nama_user  
| IDENTIFIED BY password_user;
```

Contoh :

```
SQL> create user nunit identified by nunit123;  
User created.
```

Ket: Agar user baru ini dapat log on, maka setelah di create lakukan GRANT CONNECT sebagai berikut:

```
| GRANT CONNECT, RESOURCE TO nama_user;
```

Contoh:

```
SQL> GRANT CONNECT, RESOURCE TO NUNIT;  
Grant succeeded.
```

2.1.3.1.6 Create Index

Index digunakan untuk **mempercepat pengaksesan data** pada suatu tabel. Index dapat diberikan pada satu atau lebih kolom.

Sintaks :

```
[ CREATE USER nama_user  
| IDENTIFIED BY password_user;
```

Contoh : **CREATE INDEX** nokartu **ON** member (nokartu);

2.1.3.2. Drop

2.1.3.2.1 Drop Table

Perintah ini digunakan untuk **menghapus definisi tabel** pada ORACLE yaitu semua data dan indeks yang dimiliki tabel.

Sintaks :

```
[ DROP TABLE nama_table [CASCADE CONSTRAINT];
```

Contoh : **DROP TABLE** member CASCADE CONSTRAINT;

Ket : Option **CASCADE CONSTRAINT** akan menghapus pula *constraint referensi integrity* yang terhubung.

2.1.3.2.2 Drop View

Perintah ini digunakan untuk **menghapus view** tertentu yang ada di basisdata.

Sintaks :

```
[ DROP VIEW nama_view;
```

Contoh : **DROP VIEW** view_data_transaksi;

2.1.3.2.3 Drop Sequence

Perintah ini digunakan untuk **menghapus sequence** tertentu yang ada di basisdata.

Sintaks :

```
[ DROP SEQUENCE nama_sequence;
```

Contoh : **DROP SEQUENCE** seq_transaksi;

2.1.3.2.4 Drop Synonym

Perintah ini digunakan untuk **menghapus synonym** tertentu yang ada di basisdata.

Sintaks :

```
[ DROP SEQUENCE nama_synonym;
```

Contoh : **DROP SEQUENCE** seq_transaksi;

2.1.3.2.5 Drop Index

Perintah ini digunakan untuk **menghapus index** tertentu yang ada di basisdata.

Sintaks :

```
DROP INDEX nama_index;
```

Contoh : **DROP INDEX** nokartu;

2.1.3.2.6 Drop User

Perintah ini digunakan untuk **menghapus definisi user** tertentu.

Sintaks :

```
DROP USER nama_user;
```

Contoh : **DROP USER** nunit;

2.1.3.3. Alter

2.1.3.3.1 Alter Table

Perintah ini digunakan untuk mengubah struktur tabel dan *constraint*-nya. Pada sebuah tabel ada tiga macam alter yang dapat digunakan secara terpisah ataupun digunakan bersamaan. Ketiga perintah alter tersebut yaitu:

- **ADD**

Digunakan untuk **menambah kolom(atribut) atau constraint** pada suatu tabel.

Sintaks:

```
ALTER TABLE nama_tabel  
ADD nama_atribut_yang_hendak_ditambah tipe_data constraint;
```

Contoh: **ALTER TABLE** supervisor **ADD** tgl_lahir date;

Contoh 2: **ALTER TABLE** member **ADD** (

```
Idmembership char(6),  
Constraint member_fk1 Foreign Key (idmembership) references membership  
on delete cascade);
```

- **MODIFY**

Digunakan untuk **mengubah definisi** (tipe data, *constraint*, atau ukuran *size* dari suatu tipe data) suatu kolom.

Sintaks:

```
ALTER TABLE nama_tabel  
MODIFY nama_atribut_yang_hendak_diubah tipe_data constraint;
```

Contoh: Suatu tabel **member** mempunyai sebuah kolom **telp** yang tipe datanya awalnya **char(13)**, ingin diubah menjadi tipe data **varchar2(15)**. Maka sintaksnya:

```
ALTER TABLE member MODIFY telp varchar2(15);
```

- **DROP**

Digunakan untuk **menghapus kolom(atribut) atau constraint** pada suatu tabel.

Sintaks:

```
ALTER TABLE nama_tabel  
DROP [COLUMN|CONSTRAINT] nama_atribut_yang_hendak_dihapus constraint;
```

Contoh: **ALTER TABLE** member
DROP COLUMN telp; menghapus kolom **telp**

Contoh 2: **ALTER TABLE** member
DROP CONSTRAINT member_fk1; menghapus foreign key

- **ENABLE/DISABLE**

Digunakan untuk mengaktifkan/menonaktifkan *constraint*.

```
| ALTER TABLE nama_table ENABLE CONSTRAINT nama_constraint;  
| ALTER TABLE nama_table DISABLE CONSTRAINT nama_constraint [CASCADE];
```

Contoh : **ALTER TABLE** member **ENABLE CONSTRAINT** member_pk;

Contoh : **ALTER TABLE** member **DISABLE CONSTRAINT** member_pk **CASCADE**;

Ket: Perintah **CASCADE** digunakan untuk men-disable *dependent integrity constraint*.

- **Alter View**

Perintah ini digunakan untuk **mengkompilasi ulang** sebuah *view*.

Sintaks:

```
| ALTER VIEW nama_view COMPILE;
```

Contoh: **ALTER VIEW** view_data_transaksi **COMPILE**

- **Alter Sequence**

Perintah ini digunakan untuk **memodifikasi karakteristik sequence** baik itu nilai *increment*, nilai minimum, nilai maksimum, dan lain- lain.

Sintaks:

```
| ALTER SEQUENCE nama-sequence  
| [INCREMENT BY integer]  
| [MAXVALUE integer | NOMAXVALUE]  
| [MINVALUE integer | NOMINVALUE]  
| [CYCLE | NO CYCLE];
```

Contoh: **ALTER SEQUENCE** seq_transaksi
Increment by 2
Maxvalue 1000
Minvalue 0
Cycle

Ket: Statement **START WITH** tidak dapat di-alter. Untuk statement **CYCLE** harus ada inisialisasi **MAXVALUE**.

- **Alter User**

Perintah ini digunakan untuk **mengubah password** seorang *user*.

Sintaks:

```
| ALTER USER nama-user  
| IDENTIFIED BY password;
```

Contoh: **ALTER USER** nunit
IDENTIFIED BY informaticslab;

2.1.3.3.2 Rename

Perintah ini digunakan untuk mengubah nama tabel, view, sequence dan synonym.

Sintaks:

```
| RENAME nama_lama TO nama_baru;
```

Contoh: **RENAME** member **TO** pelanggan;

2.1.3.3.3 Truncate

Oracle menyediakan **TRUNCATE** statement untuk menghapus seluruh data dari suatu tabel tanpa menghapus tabel yang ada (**yang dihapus isi tabelnya saja, tabel menjadi kosong**).

Sintaks: **TRUNCATE TABLE [schema.] nama_table;**

Contoh: **TRUNCATE TABLE member;**

2.1.3.3.4 Comment

User dapat menambahkan komentar terhadap tabel/ view. Berikut adalah perintah *comment*:

Sintaks: **COMMENT ON TABLE [schema.] nama_table IS 'isi komentar'**

Contoh: **COMMENT ON TABLE membership IS 'berisi jenis jenis keanggotaan yang tersedia dengan besaran diskon tertentu';**

2.1.3.3.5 Constraint

Jika diperhatikan dengan seksama, pada pendefenisian nama atribut terdapat kata kunci **constraint**. **Constraint** adalah pemakaian aturan yang dilakukan pada level tabel. Terdapat beberapa *constraint* yang dapat digunakan yaitu:

A. Primary key

Secara *Implisit Primary key* membentuk keunikan. **Hanya ada satu primary key yang diperbolehkan** untuk setiap tabel. Secara implisit Primary key adalah **NOT NULL**. Primary key dapat dikatakan juga sebagai *constraint Unique* yang bernilai NOT NULL. Pada sebuah Primary Key tidak boleh diberikan *constraint Unique* tetapi diperbolehkan menggunakan **NOT NULL**.

Contoh1:

```
CREATE TABLE SUPERVISOR(
    nip char(6) not null CONSTRAINT supervisor_pk
PRIMARY KEY,
    nmsupervisor varchar2(20),
    thn_masuk number
);
```

Contoh2:

```
CREATE TABLE SUPERVISOR(
    nip char(6) not null enable,
    nmsupervisor varchar2(20),
    thn_masuk number,
    CONSTRAINT supervisor_pk PRIMARY KEY (nip)
);
```

Penempatan

B. Foreign Key

Digunakan sebagai hubungan antar *primary* atau *unique key* lain. Biasanya sebuah **foreign key** pada suatu tabel merupakan **primary key** tabel yang lain. *Constraint* ini bertujuan untuk menetapkan suatu kolom atau kombinasi dari beberapa kolom menjadi *foreign key* dari sebuah tabel. *Foreign key* sering disebut sebagai *referential integrity constraint*. Kolom atau kombinasi kolom yang menjadi *Foreign Key* pada tabel anak mengacu ke kolom atau kombinasi kolom yang menjadi *Primary key* atau *Unique key* pada tabel induk.

Contoh:

```
CREATE TABLE PRODUK (
    idproduk char(6) not null,
    nmproduk varchar2(20),
    stok number,
    harga number,
    detail varchar2(50),
    spesifikasi varchar2(50),
    idkategori char(6),
    CONSTRAINT produk_pk PRIMARY KEY (idproduk) ENABLE,
    CONSTRAINT produk_fk FOREIGN KEY (idkategori)
    REFERENCES kategori (idkategori)
    ON DELETE CASCADE
);
```

Ket:

On Delete Cascade untuk **membentuk keterkaitan** antara tabel yang terhubung (dalam kasus ini tabel **produk** dan **kategori**). Jika suatu baris pada tabel **kategori** dihapus, maka kolom **produk** yang terhubung juga dihapus.

C. Null / Not Null

Jika sebuah kolom pada *database* tidak boleh kosong, maka *constraint NOT NULL* diberikan pada kolom tersebut. Pada saat tabel diisi dengan data (misalnya melalui instruksi *INSERT*), maka kolom tersebut harus diisi. Jika kolom tidak diisi, sistem akan menolaknya.

Constraint NULL berarti bahwa kolom dapat diisi dan dapat dikosongkan.

Contoh:

```
CREATE TABLE SUPERVISOR(
    nip char(6) NOT NULL CONSTRAINT supervisor_pk primary key,
    nmsupervisor varchar2(20) NOT NULL,
    thn_masuk number
);
```

D. Unique

Merupakan *constraint* yang memiliki sifat yang hampir sama dengan **primary key**, yaitu harus memiliki **nilai yang berbeda untuk setiap baris pada satu kolom yang sama**. Hanya saja, *constraint unique* mengijinkan adanya nilai **NULL(kosong)**.

E. Check

Constraint *Check* digunakan untuk memeriksa nilai suatu kolom apakah memenuhi suatu kriteria yang diinginkan. Kriteria-kriteria tersebut dapat berupa **Perbandingan, range, NULL, LIKE, EXISTS**.

Constraint check **tidak dapat digunakan** pada:

Acuan *pseudo-columns* seperti SYSDATE, UID, dan lain lain atau *Query* yang mengacu pada nilai pada kolom lain.

Contoh :

```
CREATE TABLE PRODUK (
    idproduk char(6) not null,
    nmproduk varchar2(20),
    stok number,
    harga number,
    detail varchar2(50),
    spesifikasi varchar2(50),
    idkategori char(6),
    CONSTRAINT produk_pk PRIMARY KEY (idproduk) ENABLE,
```

```

CONSTRAINT produk_fk FOREIGN KEY (idkategori) REFERENCES
kategori (idkategori)ON DELETE CASCADE
CONSTRAINT produk_ck CHECK (stok<50)
);

```

2.1.4 Data Manipulation Language (DML)

2.1.4.1. Pengenalan

Data Manipulation Language merupakan bahasa yang memungkinkan pengguna untuk mengakses dan mengubah data yang sesuai dengan model datanya selain itu DML juga merupakan konsep yang menerangkan bagaimana menambah, mengubah dan menghapus baris tabel.

2.1.4.2. Perintah-perintah DML

2.1.4.2.1 Insert

Berfungsi untuk **menambahkan** baris baru ke tabel

Sintaks:

```

INSERT INTO <nama_tabel> [(nama_kolom1, nama_kolom2, ....)]
Values (nilai1, nilai2, ...);

INSERT INTO <nama_tabel> [(nama_kolom1, nama_kolom2,...)] <query>;

```

Contoh:

```

SQL> INSERT INTO SUPERVISOR VALUES('SUP006','Pambudi',2014);
1 row created.

```

2.1.4.2.2 Update

Berfungsi untuk **memodifikasi** nilai pada baris tabel.

Sintaks:

```

UPDATE <nama_tabel>
Set <nama_kolom1>=<nilai_ekspresi1>,
<nama_kolom2>=<nilai_ekspresi2>,
...
{Where <kondisi>};

```

Contoh:

```

SQL> UPDATE supervisor SET thn_masuk=2013 WHERE nip='SUP006';
1 row updated.

```

2.1.4.2.3 Delete

Berfungsi untuk **menghapus** baris tunggal atau lebih dari satu baris berdasarkan kondisi tertentu.

Sintaks:

```

DELETE FROM <nama_tabel>
{WHERE <kondisi>};

```

Contoh :

```

SQL> DELETE FROM supervisor WHERE nip='SUP006';
1 row deleted.

```

Jika klausula *where* tidak digunakan maka semua data pada tabel akan terhapus. Sebagai sintaks alternatif untuk menghapus semua data pada table digunakan perintah TRUNCATE.

Sintaks:

```
TRUNCATE TABLE <table> [reusage storage];
```

2.1.4.2.4 Merge

Berfungsi untuk melakukan *update* maupun *insert* ke suatu tabel tertentu berdasarkan kondisi dari tabel lain. Dengan perintah merge ini, data akan di-*insert*-kan apabila data tersebut tidak ada di tabel tujuan dan akan di-*update*-kan apabila data tersebut telah ada di tabel tujuan.

Sintaks:

```
MERGE INTO <nama_tabel> <nama_alias_tabel>
USING (table|view|sub_query) alias
ON (join condition)
    WHEN MATCHED THEN
        UPDATE SET [(column1=column_val1), (column2=column_val2),...]
    WHEN NOT MATCHED THEN
        INSERT (column_list)
        VALUES (column_values);
```

Contoh :

```
MERGE INTO pelanggan p
USING (select * from member) m
ON m.idmember=p.idmember
WHEN MATCHED THEN
    UPDATE SET p.password=m.password
WHEN NOT MATCHED THEN
    INSERT (p.idmember) VALUES (m.idmember);
```

Penjelasan :

Query di atas akan melakukan proses *update* ke tabel **pelanggan** apabila idmember ada yang sama dengan idmember di tabel **member**, sedangkan apabila idmember pada tabel **member** tidak ada di tabel **pelanggan** maka akan dilakukan proses *insert* ke tabel **pelanggan** sesuai dengan nilai yang terdapat pada tabel **member**.

2.1.5 View

2.1.5.1 Pengertian View

View merupakan sebuah **tabel semu / tabel logik**, dimana datanya berasal dari satu atau lebih tabel lain sebagai tabel sumber. *View* tidak memiliki indeks, terdapat batasan dalam memperbarui data dalam *view*. Ketika *view* diperbarui maka perubahannya dialurkan ke tabel dasar yang mendasari referensi untuk membuat *view*. Tabel dasar merupakan tabel dalam model data relasional yang berisikan *raw data*.

2.1.5.2 Jenis – Jenis View

2.1.5.2.1 Dynamic view

Dynamic view merupakan sebuah tabel virtual yang dibuat secara dinamis sesuai permintaan pengguna. *Dynamic view* bukan merupakan tabel sementara, melainkan tempat menyimpan definisinya dalam katalog sistem. Dan konten dari *view* diwujudkan sebagai hasil dari *query sql*

yang menggunakan *view* tersebut. Berbeda dengan *materialized view* yang dapat disimpan pada *disk* yang dapat diperbaharui sesuai interval tertentu. Pernyataan *SQL SELECT* apapun dapat digunakan untuk membuat *view*. Data yang ada disimpan pada tabel dasar, yang telah ditentukan oleh perintah *SQL CREATE TABLE*. *Dynamic view* selalu berisi nilai turunan terbaru dan karenanya lebih unggul dalam hal data *currency* daripada membuat tabel nyata sementara dari beberapa tabel dasar.

Tabel 2-1. Kelebihan dan kekurangan menggunakan dynamic view

Aspek positif	Aspek negatif
Menyederhanakan perintah query	Penggunaan dan waktu proses yang lama, karena kontennya dihitung setiap kali digunakan.
Membantu memberikan keamanan dan kerahasiaan data	Jika struktur tabel dasar dirubah, dan struktur tersebut direfensikan dalam <i>view</i> , maka tampilan harus dibuat ulang untuk mencegah kesalahan query.
Meningkatkan produktivitas programmer	
Berisi data tabel dasar terbaru	
Menggunakan ruangan penyimpanan yang kecil	
Memberikan <i>view</i> yang dapat disesuaikan untuk pengguna	
Membangun independensi data fisik	

2.1.5.2.2 Materialized view

Materialized view merupakan salinan atau replikasi data berdasarkan *query SQL* yang dibuat dengan cara yang sama seperti *dynamic view*. Namun *materialized view* diwujudkan sebagai tabel sehingga harus tetap diperhatikan untuk tersinkronisasi dengan tabel dasarnya yang terikat. Sama halnya dengan *dynamic view*, *materialized view* dapat dibangun dengan cara yang berbeda untuk berbagai tujuan. Tabelnya dapat direplikasi secara keseluruhan atau sebagian dan diperbaharui pada interval waktu yang telah ditentukan atau ketika tabel perlu untuk diakses. *Materialized view* dapat didasarkan pada *query* dari satu atau lebih tabel, sehingga dimungkinkan untuk membuat tabel ringkas berdasarkan agregasi data.

Tabel 2-2. Kelebihan dan kekurangan menggunakan materialized view

Aspek positif	Aspek negatif
Menyederhanakan perintah query	Menggunakan penyimpanan yang lebih besar daripada <i>dynamic view</i>
Membantu memberikan keamanan dan kerahasiaan data	Data harus dirawat(<i>maintained</i>) dengan baik
Meningkatkan produktivitas programmer	Mungkin tidak berisi data tabel terbaru
Memberikan <i>view</i> yang dapat disesuaikan untuk pengguna	
Membangun independensi data fisik	
Dapat meningkatkan kinerja <i>query</i> terdistribusi, terutama jika data dalam <i>view</i>	

relatif statis dan tidak sering untuk diperbaharui	
--	--

2.1.5.3 Create View

2.1.5.3.1 Dynamic view

Sintaks: Sintaks dalam membuat *view* adalah sebagai berikut :

```
CREATE VIEW nama_view AS  
<query expression>
```

Query expression adalah ekspresi *query* apapun yang akan menjadi definisi dari sebuah *view*.

Contoh :

```
CREATE VIEW OutstandingStudent AS  
SELECT NAME, COURSE, GRADE  
FROM STUDENT  
WHERE GRADE>B;
```

2.1.5.3.2 Materialized view

Sintaks :

```
CREATE TABLE nama_view AS  
<query expression>
```

Query expression adalah ekspresi *query* apapun yang akan menjadi definisi dari sebuah *view*.

Contoh :

```
CREATE TABLE OutstandingStudent AS  
(SELECT NAME, COURSE, GRADE  
FROM STUDENT  
WHERE GRADE>B);
```

2.1.6 Studi Kasus

Perhatikan studi kasus berikut ini!

Sebuah bioskop memiliki beberapa ruang *theater*. Setiap *theater* hanya memiliki 1 tipe kelas, 1 kapasitas dan 1 harga tertentu. Masing-masing ruang *theater* memiliki kapasitas kursi yang berbeda. Kursi –kursi pada setiap theater memiliki label yang menunjukkan posisi baris dan nomor. Seluruh kursi pada bioskop tersebut didata sebagai inventaris.

Setiap film yang diputar akan memiliki jadwal, dimana pada jadwal tersebut akan dijelaskan film yg diputar, jam penayangan, *theater* yang akan menayangkan dan lain-lain.

Business Rule:

- 1 film bisa diputar pada lebih dari 1 *theater*.
- 1 *theater* bisa memutar lebih dari 1 film.
- Masing-masing film memiliki detail informasi terkait film tersebut

Seluruh data-data tersebut harus dapat tersimpan dalam *database* dan bisa di-*inquiry* untuk keperluan pelaporan.

Sebagai gambaran, berikut beberapa contoh data yang ingin disimpan :

- Contoh 1:
Theater 1 memiliki 10 kursi dengan rincian: 5 baris (A,B,C,D,E). masing-masing baris diisi 2 nomor (1,2)
- Contoh 2:
Theater 2 memiliki 20 kursi dengan rincian: 5 baris (A,B,C,D,E). Masing-masing baris diisi 4 nomor (1,2,3,4)
- Contoh 3:

Spider-Man: Far From Home(2019)



Sinopsis:

Spider-Man Far From Home menjadi film marvel pertama setelah Avengers: Endgame. Film ini menjadi sekuel lanjutan dari fil Spider Man Homecoming yang dirilis 5 Juli 2017 silam. Jon Watts didapuk menjadi sutradara dalam film ini, sedangkan skenarionya ditulis oleh Chris McKenna dan Erik Sommers. Aktor Tom Holland masih didapuk sebagai pemeran utama dalam film ini.

Dalam Spider-Man: Far From Home sebagian akan berurusan dengan bagaimana Peter menangani Decimation dan fakta bahwa Tony Stark sekarang sudah mati. Seperti yang ditunjukkan trailer terbaru, Peter akan dihadapkan dengan kematian Tony di setiap kesempatan. Dia tidak akan punya banyak waktu untuk bersedih, karena Spider-Man akan direkrut oleh Nick Fury untuk menangani beberapa urusan yang mendesak. Ternyata, Spider-Man direkrut untuk bertarung melawan makhluk kuno yang dikenal sebagai Elementals, yang meliputi Hydron, Hellfire, Magnum, dan Zephyr. Saat bertarung dengan Elementals, Spider-Man akan dibantu oleh Mysterio, karakter pemegang sihir yang dimainkan oleh Jake Gyllenhaal. Tetapi karakter baru Gyllenhaal yang misterius mungkin memiliki motif tersembunyi.

Siapakah Mysterio? Dia adalah karakter yang memiliki pengetahuan dan keterampilan yang hebat untuk menciptakan ilusi yang meyakinkan dan efek khusus. Beberapa orang berspekulasi bahwa Mysterio adalah seorang mistikus seperti Doctor Strange, meskipun dia tidak setinggi itu. Dalam kasus ini Mysterio gagal sebagai seorang mistikus tetapi masih berpegang teguh pada harapan bahwa dia bisa menjadi pahlawan. Jadi dia menggunakan ilusi, yang merupakan keahlian terbaiknya. Tentu saja, Mysterio bisa saja menggunakan gadget yang menunjang penampilannya. Intinya adalah, dia mungkin bertindak seperti pahlawan, tetapi dia akan terungkap sebagai penjahat.

MPAA : PG-13

Durasi : 2h 9min

Film tersebut akan diputar di bioskop XXI pada theater 1 dan 2. Pada theater 1, akan diputar pada tanggal 3 Juli 2019 s.d 20 Juli 2019, jam 12.30 – 14.30; 16.00 – 18.00; 19.30 – 21.30. Pada theater 2,

akan diputar pada tanggal 3 Juli 2019 s.d 15 Juli 2019, jam 13.00 – 15.00; 16.00 – 18.00; 19.00 – 21.00.

2.1.7. Tutorial

DDL(*Data definition Language*)

1. Pastikan SQLPLUS oracle sudah berjalan dengan baik.
2. Tentukan tipe data dan ukuran untuk masing-masing tabel.

Nama Tabel:		
Atribut	Tipe data	Ukuran

Nama Tabel:		
Atribut	Tipe data	Ukuran

Nama Tabel:		
Atribut	Tipe data	Ukuran

Nama Tabel:		
Atribut	Tipe data	Ukuran

Nama Tabel:		
Atribut	Tipe data	Ukuran

Nama Tabel:		
Atribut	Tipe data	Ukuran

3. Create semua tabel yang ada pada skema relasi. Mulai dengan *table* yang tidak memiliki *foreign key* atau yang tidak memiliki atribut yang bergantung dengan tabel lain.

Tabel tanpa foreign key/Tabel Master	Tabel memiliki foreign key
--------------------------------------	----------------------------

DML (Data Manipulation language)

1. Lakukan *insert* data film sebagai berikut !

Id_film	judul	Durasi (menit)	tahun	sinopsis
F0101	Keluarga Cemara	110	2019	Fokus cerita ini masih soal, Emak serta kedua anak mereka, Euis dan Ara . Persis dengan cerita di sinetronnya, film ini mengisahkan tentang bagaimana perjalanan hidup keluarga Abah yang semula nyaman dan mapan lalu kemudian mendadak bangkrut. Abah pun mengajak keluarganya pindah ke rumah warisan yang cukup jauh dari kota. Dengan segala keterbatasan dan kekurangan yang ia memiliki, Abah berusaha menjadi kepala keluarga serta ayah yang baik untuk anak-anaknya. Untung saja, Emak tak pernah lelah mendampinginya dan berusaha menjadi sandaran di saat mereka sedih.
F0102	Habibie Ainun 3	96	2019	Habibie & Ainun 3 adalah sebuah film Indonesia tahun 2019 yang disutradarai oleh Hanung Bramantyo dan diproduksi oleh Manoj Punjabi (MD Pictures). Film ini adalah film ketiga dari seri film Habibie & Ainun. Bila "Rudy Habibie" merupakan prequel dari kisah Habibie muda, maka "Habibie & Ainun 3" ini adalah prequel dari kisah Ainun muda. Film ini dijadwalkan rilis pada 19 Desember 2019. Hasri Ainun Besari diperankan oleh Maudy Ayunda menggantikan Bunga Citra Lestari pada film pertama, karena film ini akan lebih menekankan pada kisah Ainun di masa muda.
F0103	Taufiq	105	2019	Kisah penjalanan hidup politisi yang juga suami Mantan Presiden RI, Megawati Soekarnoputri, Taufiq Kiemas diangkat dalam sebuah film tayang 14 Maret 2019. Film yang diangkat dari kisah nyata itu disutradarai sekaligus skenarionya ditulis oleh Ismail Basbeth. Perjalanan hidup membawanya ke Jakarta dan bertemu langsung dengan Soekarno, menjalin persahabatan dengan Guntur Soekarnoputra serta bertemu Megawati Soekarnoputri untuk pertama kalinya. Taufiq akan

Id_film	judul	Durasi (menit)	tahun	sinopsis
				memahami arti dari persahabatan, keluarga, cinta dan negara ketika dia menjalani ujian hidup sebagai seorang nasionalis. Setelah itu ia dijebloskan ke penjara bersama sahabatnya dituduh sebagai anggota Komunis. Dalam penjara Taufiq banyak bertemu orang-orang hebat. Dari situ lah perjalanan seorang lelaki yang menantang badai.
F0104	Buya Hamka	110	2019	Buya Hamka adalah film drama biografi Indonesia tentang Abdul Malik Karim Amrullah atau Hamka, seorang ulama Indonesia yang dikenal sebagai penulis, pujangga, dan politisi. Buya Hamka akan mengisahkan kehidupan Hamka sejak lahir sampai meninggal dunia. Sebelum menjadi sosok yang dikenal banyak orang, Hamka telah melewati beberapa perubahan, mulai dari kanak-kanak, remaja, hingga berkeluarga. Sebagai ulama, film ini menyoroti bagaimana cara Hamka menyampaikan dakwahnya secara santun. Selain itu, Buya Hamka menyoroti aspek humanis Hamka dan prosesnya menggapai semua pencapaian.

2. Lakukan insert data teater sebagai berikut !

Nomor_teater	Kelas	harga	kapasitas
Teater 1	Reguler	30000	50
Teater 2	Sweetbox	100000	50
Teater 3	4D	75000	50
Teater 4	Velvet	80000	50
Teater 5	3D	50000	50

3. Lakukan insert data Member sebagai berikut !

Id_member	Nama_member	No_hp	email	Tgl_lahir
MM0111	Anto	085267656789	Anto2016@gmail.com	1989-07-21
MM0112	Budi	081367589632	Budi2016@gmail.com	1985-01-03
MM0113	Ari	081267867543	Ari2016@gmail.com	1983-11-24
MM0114	Rahmi	085267935678	Rahmi2016@gmail.com	1981-09-01
MM0115	Fahmi	085767298908	Fahmi2016@gmail.com	1986-07-15
MM0116	Rusli	085643755398	Rusli2016@gmail.com	1988-02-26
MM0117	Doni	081398426789	Doni2016@gmail.com	1986-04-17
MM0118	Tati	085245289074	Tati2016@gmail.com	1985-06-22
MM0119	Dono	081287234567	Dono2016@gmail.com	1990-07-03
MM0120	Joko	081223670942	Joko2016@gmail.com	1988-07-19

4. Lakukan insert data jadwalTayang sebagai berikut !

Id_jadwalTayang	Id_film	Nomor_teater	Periode_start	Periode_end
JT001	F0101	Teater 1	01/07/2019	07/07/2019
JT002	F0101	Teater 1	01/07/2019	07/07/2019

JT003	F0102	Teater 1	01/07/2019	07/07/2019
JT004	F0101	Teater 2	01/07/2019	07/07/2019
JT005	F0103	Teater 2	01/07/2019	07/07/2019
JT006	F0102	Teater 2	01/07/2019	07/07/2019
JT007	F0104	Teater 2	01/07/2019	07/07/2019
JT008	F0102	Teater 3	01/07/2019	07/07/2019
JT009	F0101	Teater 1	01/07/2019	07/07/2019
JT010	F0102	Teater 1	01/07/2019	07/07/2019

5. Lakukan insert data kursi sebagai berikut !

No_inventori	Nomor_teater	No_kursi
IN1001	Teater 1	A1
IN1002	Teater 1	A2
IN1003	Teater 1	A3
IN1004	Teater 1	A4
IN1005	Teater 1	A5
IN1006	Teater 2	A6
IN1007	Teater 2	A7
IN1008	Teater 2	A8
IN1009	Teater 2	B1
IN1010	Teater 2	B2
IN1011	Teater 3	C1
IN1012	Teater 3	C2

6. Lakukan insert data order sebagai berikut !

Id_order	Id_member	Id_jadwalTayang	No_inventori	Tgl	Status
P10001	MM0111	JT001	IN1010	15/06/2019	pesan
P10002	MM0112	JT006	IN1009	16/06/2019	pesan
P10003	MM0113	JT008	IN1008	17/06/2019	pesan
P10004	MM0114	JT001	IN1007	18/06/2019	pesan
P10005	MM0115	JT006	IN1006	19/06/2019	pesan
P10006	MM0116	JT001	IN1001	20/06/2019	bayar
P10007	MM0117	JT008	IN1011	21/06/2019	pesan
P10008	MM0118	JT001	IN1003	22/06/2019	pesan
P10009	MM0119	JT006	IN1004	23/06/2019	pesan
P10010	MM0120	JT008	IN1005	24/06/2019	checkin

7. Lakukan Update data member sebagai berikut !

Nomer HP member dengan id MM0113 an Ari berubah menjadi 08098933333

Email member dengan id MM0117 an Doni menjadi doni_inod@gmail.com

Teater 5(3D) harganya berubah menjadi 100000.

8. Lakukan penghapusan data Film dengan judul “Buya Hamka” dan teater 3 (4D) dari tabel yang sudah di buat.

Modul 3. Scripting SQL Basic Query – Pemrosesan Satu Tabel

Tujuan Praktikum

1. Memahami konsep dasar transaksi
2. Mengetahui dan memahami perintah-perintah pada transaksi
3. Mengetahui dan memahami definisi dan fungsi query

3.1. Ringkasan Materi

3.1.1. SQL, SQL*Plus dan PL / SQL

Structured Query Language (SQL) adalah bahasa *query non-procedural* level tinggi yang paling sering digunakan. SQL merupakan bahasa yang digunakan untuk mengakses Basis Data Relasional termasuk Oracle.

PL/SQL merupakan bahasa *procedural* pada Oracle untuk menulis aplikasi logik dan untuk memanipulasi data diluar basis data.

Bentuk dasar dari SQL :

```
-----  
| Select A1, A2,..., From R1, R2, ...  
| Where P;  
-----
```

Keterangan :

A : atribut

R : relasi

P : predikat

*keluaran dari query SQL adalah sebuah relasi

3.1.2. Query Dasar

Query merupakan *statement* dalam SQL yang berfungsi untuk mendapatkan atau mengambil data dari *database* (satu atau beberapa tabel/view) berdasarkan kriteria-kriteria tertentu yang diberikan^[3]. Sebuah *query* selalu diawali dengan *SELECT statement*. Secara umum sintaks *query* sederhana dapat dituliskan seperti di bawah ini^[3].

Sintaks *query* sederhana:

```
-----  
| SELECT [ALL|DISTINCT] (nama_kolom1, nama_kolom2, ....)  
| FROM (nama_tabel_sumber1, nama_tabel_sumber2, .... )  
| {WHERE (kondisi});  
-----
```

Keterangan :

- **SELECT:** Untuk menspesifikasi nama-nama kolom yang akan ditampilkan. Nama-nama kolomnya dituliskan setelah klausa ini.
- **DISTINCT:** Untuk menampilkan *record-record* hasil *query* yang nilai atau *values*-nya berbeda. Tujuan penggunaan klausa ini untuk menghilangkan *redundancy* hasil *query*.
- **ALL:** Untuk menampilkan seluruh *record* meskipun ada beberapa *record* yang nilainya sama. Secara default setiap *statement* *SELECT* menggunakan klausa *ALL* di dalamnya.

- **FROM:** Mendefinisikan nama tabel yang mengandung kolom yang terdaftar dalam *SELECT*
- **WHERE:** Untuk mendefinisikan kondisi atau kriteria menyaring data yang diambil dari tabel sumber. Terdiri dari 3 elemen, yaitu nama kolom, kondisi perbandingan dan nama konstanta atau daftar nilai.

3.1.3. Contoh Query

1. Menampilkan semua data (gunakan tanda ‘*’) dari tabel barang

Query :

```
SELECT * FROM produk;
```

Hasil :

IDPROD	NMPRODUK	STOK	HARGA
000001	TU Sharp 32	18	2450000
000002	Lampu gantung	10	2500000
000003	TU Sharp 23	20	1900000
000004	TU LG 32	19	2500000
000005	L.Es panasonic	12	1250000
000006	L.Es panasonic	20	2000000
000007	Yongma 181	11	2500000

Gambar 3-1. Output query select

2. Menampilkan beberapa kolom yang spesifik (penggunaan klausa *WHERE*)

Contoh: Menampilkan kolom nama barang dan jumlah stok dari tabel barang yang jumlah stocknya kurang dari 15 (pengkondisian).

Query :

```
SELECT nmProduk, stok, harga
FROM produk
WHERE idproduk='000001';
```

Hasil :

SQL> SELECT nmProduk, stok, harga FROM produk WHERE stok < 15;		
NMPRODUK	STOK	HARGA
Lampu gantung	10	250000
L.Es panasonic	12	1250000
Yongma 181	11	250000

Gambar 3-2. Output query select dengan where

Keterangan :

- a. Jika pengkondisian bertipe data string atau date maka gunakan tanda petik satu ('_').
 - b. Nilai pada tipe data string harus case sensitive.
 - c. Nilai tipe data harus format *sensitive*. Default dari format date adalah DD-MM-YYYY.
3. Menampilkan nama barang yang ada di tabel barang (penggunaan klausa *DISTINCT*)

Dengan klausa DISTINCT maka nama kolom yang di DISTINCT hanya akan muncul 1x saja. Klasa ini berguna untuk melihat ragam nilai yang ada pada suatu kolom.

Query :

```
SELECT DISTINCT nmMember
FROM member ;
```

4. Menampilkan jumlah member berdasarkan tipe member (penggunaan klausanya GROUP BY)

Query :

```
SELECT jenis, count(*) jml
FROM membership join member using(idmembership)
GROUP BY (jenis);
```

Hasil :

```
SQL> SELECT jenis, count(*) jml
2   FROM membership join member using(idmembership)
3   GROUP BY (jenis);
```

JENIS	JML
Silver	1
Gold	2
Platinum	2

Gambar 3-3. Output query select dengan group by

5. Menampilkan semua nama produk terurut DESCENDING berdasarkan harga (penggunaan klausanya ORDER BY), default dari order by adalah ASCENDING, jadi klausanya ascending boleh ditulis atau tidak .

Query :

```
SELECT *
FROM produk
ORDER BY harga DESC;
```

Hasil :

```
SQL> SELECT * FROM produk
2 ORDER BY harga DESC;

IDPROD NMPRODUK           STOK      HARGA
----- ----- -----
000004 TU LG 32            19       2500000
000001 TU Sharp 32          18       2450000
000006 L.Es panasonic       20       2000000
000003 TU Sharp 23          20       1900000
000005 L.Es panasonic       12       1250000
000002 Lampu gantung        10       250000
000007 Yongma 181           11       250000
```

Gambar 3-4. Output query select dengan order by

Query Tambahan

6. Untuk melihat daftar *table* yang telah kita masukkan di dalam *database*

```
SELECT * FROM TAB;
```

Hasil :

```
SQL> select*from tab;

TNAME                TABTYPE CLUSTERID
----- ----- -----
BARANG                TABLE
PEGAWAI                TABLE
CUSTOMER               TABLE
KEANGGOTAAN             TABLE
KARTU_ANGGOTA             TABLE
PEMBELIAN               TABLE
PEMBELIANS_DETIL             TABLE

7 rows selected.
```

Gambar 3-5. Output query select dengan tab

7. Untuk melihat daftar atribut yang terdapat pada suatu *table*

```
DESC [Nama Tabel];
```

Hasil :

```
SQL> desc produk;
Name
-----
IDPRODUK
NMPRODUK
STOK
HARGA
DETAIL
SPESIFIKASI
IDKATEGORI
```

Gambar 3-6. Output query dengan Desc

3.1.4. Tentang Query

Beberapa hal yang perlu diperhatikan dalam penulisan *statement* dalam SQL :

1. SQL *statement* tidak *case sensitive*
2. SQL *statement* dapat dibuat dalam satu baris atau lebih
3. *Keyword* tidak dapat disingkat
4. Klausa biasanya ditempatkan di baris yang terpisah
5. Spasi digunakan untuk mempermudah

Untuk menyimpan sintaks *query* yang baru dieksekusi, maka digunakan perintah SAVE, contoh :

```
SQL> SELECT * FROM transaksi;
SQL> SAVE data_trans;
```

Untuk memanggil sintaks *query* tersebut dapat dengan perintah START atau tanda '@', contoh :

```
SQL> START data_trans;
Atau
SQL> @data_trans;
```

3.1.5. Klausa SELECT

Ada kemungkinan untuk menyisipkan item lain pada klausa SELECT yaitu :

1. Ekspresi Aritmatik

Terdiri atas : * ; / ; + ; -

Contoh :

```
SELECT nmProduk, stok+2
FROM produk
WHERE idProduk ='000001';
```

Keterangan :

- a. Perkalian dan pembagian didahulukan daripada penjumlahan dan pengurangan
- b. Operator dengan prioritas yang sama (misal perkalian dan pembagian) akan dieksekusi dari operator yang paling kiri (pertama ditulis)
- c. Tanda kurung '(' digunakan untuk menandakan bahwa *statement* di dalamnya harus dieksekusi terlebih dahulu.

2. Kolom Alias

Digunakan untuk mengganti *heading* dari hasil *query* atau nama tabel. Kolom alias ditulis **setelah** **klausa SELECT**. Secara default, alias ditampilkan dengan huruf kapital dan tidak ada spasi. Dibutuhkan tanda kutip dua ("") jika nama alias mengandung spasi atau karakter khusus atau jika kasusnya *sensitive*. Dapat juga menggunakan keyword **AS**.^[3]

Contoh :

```
SELECT nmMember Nama, alamat AS alamat, telp " nomor telepon"  
FROM member;
```

3. Kolom Konkat

Operator konkat (||) atau penggabungan digunakan untuk menghubungkan suatu kolom dengan kolom lain, ekspresi aritmatik atau nilai konstan untuk membentuk ekspresi karakter.

Contoh :

```
SELECT nmMember ||' '|| alamat Data Member  
FROM member ;
```

4. Literal

Merupakan karakter, ekspresi atau bilangan yang terdapat pada klausa SELECT namun bukan merupakan salah satu *field* atau kolom pada tabel yang digunakan (tabel setelah klausa FROM) dan ingin ditampilkan pada setiap baris hasil *query*.

Contoh :

```
SELECT nmMember, 'memiliki alamat di',alamat FROM member ;
```

3.1.6. Operator-Operator Dalam Oracle

1. Operator Logika

Operator logika AND dan OR digunakan untuk membuat sebuah kondisi yang lebih kompleks yang melibatkan beberapa kondisi yang lebih sederhana. Sedangkan operator NOT dipakai untuk menyatakan negasi dari suatu kondisi. Bentuk umum penggunaan operator AND dan OR adalah Sintaks :

WHERE kondisi1 [**AND|OR**] kondisi2 [**AND|OR**] kondisi3 ...

Contoh : menampilkan data supervisor yang namanya berawalan 'A' atau 'Z'

```
SELECT nmsupervisor, thn_masuk FROM supervisor  
WHERE nmsupervisor LIKE 'A%' OR nmsupervisor LIKE 'Z%';
```

2. Operator Pembanding

a. Operator dengan Lambang Aritmatik

Tabel 9-1. Operator aritmatik

Operator	Definisi
=	Sama dengan
>	Lebih besar dari
<	Lebih kecil dari
>=	Lebih besar atau sama dengan
<=	Lebih kecil atau sama dengan
<> atau !=	Tidak sama dengan

Contoh : Menampilkan nama barang yang jumlah stocknya kurang dari 100.

```
SELECT nmproduk FROM produk WHERE stok < 15;
```

b. **IN dan NOT IN**

Operator IN dan NOT IN digunakan untuk membandingkan sebuah nilai terhadap nilai di dalam (IN) atau di luar (NOT IN) sebuah *list*. Bentuk umumnya adalah :

Sintaks :

```
WHERE nama_kolom IN (nilai1,nilai2,nilai3,...)
```

```
WHERE nama_kolom NOT IN (nilai1,nilai2,nilai3,...)
```

Contoh : menampilkan nama barang yang jumlah stocknya selain 10 dan 20.

```
SELECT nmproduk FROM produk WHERE stok NOT IN (10,20);
```

Selain untuk yang bernilai integer atau bilangan bisa juga dengan membandingkan tipe string.

Contoh: menampilkan besar diskon yang jenis membernya selain platinum dan gold.

```
SELECT diskon FROM membership
```

```
WHERE jenis NOT IN ('Platinum','Gold');
```

c. **BETWEEN dan NOT BETWEEN**

Operator ini digunakan untuk membandingkan suatu nilai terhadap suatu rentang nilai (*BETWEEN*) atau di luar rentang (*NOT BETWEEN*). Operator ini memerlukan dua parameter sebagai nilai batas awal dan nilai batas akhir.

Sintaks:

```
WHERE nama_kolom BETWEEN nilai_awal AND nilai_akhir
```

```
WHERE nama_kolom NOT BETWEEN nilai_awal AND nilai_akhir
```

Contoh: menampilkan semua nama barang yang memiliki jumlah stock antara 10-20.

```
SELECT nmproduk FROM produk WHERE stok BETWEEN (10,20);
```

d. **NULL**

Data *NULL* tidak dapat digunakan secara langsung dalam perhitungan karena nilai *NULL* bukan berarti nol, melainkan tidak diketahui atau tidak ada. Oleh karena itu data *NULL* tidak bisa dibandingkan terhadap suatu nilai yang actual. Operator *IS NULL* digunakan untuk mengenali data *NULL*.

Sintaks:

```
WHERE nama_kolom IS NULL
```

```
WHERE nama_kolom IS NOT NULL
```

Contoh: menampilkan semua data dari tabel member dari field yang teleponnya *NULL*

```
SELECT * FROM member
```

```
WHERE telp IS NULL;
```

e. *LIKE* dan *NOT LIKE*

Operator *LIKE* dan *NOT LIKE* digunakan untuk mencari suatu nilai bertipe string dengan membandingkan susunan karakternya.

Sintaks:

WHERE *nama_kolom* **LIKE** *nilai_pembanding* [**ESCAPE** *char*]
WHERE *nama_kolom* **NOT LIKE** *nilai_pembanding* [**ESCAPE** *char*]

Terdapat dua WILDCARD yang dapat digunakan untuk mengabaikan karakter-karakter yang tidak perlu dibandingkan , keduanya adalah : ^[3]

- ❖ Tanda ‘_’ mewakili satu buah karakter sembarang
- ❖ Tanda ‘%’ mewakili nol atau lebih karakter sembarang

Contoh: menampilkan nama *customer* yang diawali dengan huruf ‘S’

```
SELECT nmMember FROM member  
WHERE nmMember LIKE ‘S%’;
```

Contoh : menampilkan nama *customer* yang terdiri dari 4 karakter

```
SELECT nmMember FROM member  
WHERE nmMember LIKE ‘_____’;
```

Bagaimana jika string yang ingin dibandingkan mengandung karakter ‘%’ atau ‘_’ ? untuk mengatasinya, oracle menyediakan kata kunci **ESCAPE** sehingga kedua *wildcard* tersebut tetap digunakan sebagai karakter pembanding. Dalam penggunaannya, **ESCAPE** diikuti oleh sebuah karakter tunggal sembarang yang menjadi *escape character*. Tanda ‘%’ dan ‘_’ yang mengikuti *escape character* tersebut akan dianggap sebagai karakter pembanding, bukan sebagai *wildcard*.

Contoh : menampilkan email member yang emailnya diawali huruf ‘M’ dan terdapat karakter ‘_’

```
SELECT nmMember FROM member  
WHERE nmMember LIKE ‘M%＼_%’ ESCAPE ‘＼’;
```

KET : Escape karakter \ pada ‘C%＼_%’ di atas berfungsi untuk membuat karakter ‘_’ di belakangnya menjadi karakter pembanding, bukan sebagai *wildcard*. Sembarang karakter-pun boleh menjadi *escape character*.

3. Derajat Operator

Tabel 3-2. Derajat operator

Derajat	Operator
1	NOT
2	+ - (Unary Operator)
3	* /
4	+ -
5	= <> != ^= < > <= >= LIKE BETWEEN IN IS NULL
6	AND
7	OR

3.1.7. Fungsi-Fungsi

1. Fungsi Karakter dan String

Tabel 3-3. Fungsi Manipulasi Kasus

Fungsi Manipulasi Kasus	
<i>LOWER (column/expression)</i>	Mengkonversikan <i>column/expression</i> menjadi huruf kecil semua
<i>UPPER (column/expression)</i>	Mengkonversikan <i>column/expression</i> menjadi huruf kapital semua
<i>INITCAP (column/expression)</i>	Mengkonversikan huruf pertama dari setiap kata menjadi huruf kapital

Fungsi String	Arti
<i>CONCAT (column1/expression1, column2/expression2)</i>	Menggabungkan hanya dua string, ekivalen dengan ()
<i>LENGTH (column/expression)</i>	Menghasilkan panjang suatu string
<i>SUBSTR (column/expression,m,[n])</i>	Memotong string dimulai dari karakter ke m sebanyak n karakter. Jika n tidak dituliskan maka pemotongan akan dilakukan sampai akhir string.
<i>LPAD(column/expression,n,['string2'])</i> <i>RPAD(column/expression,n,['string2'])</i>	Menambah di sebelah kiri (LPAD) atau sebelah kanan (RPAD) dari ekspresi dengan <i>string2</i> hingga mencapai n karakter. Bila <i>string2</i> tidak dituliskan maka akan ditambah dengan spasi. Bila n < panjang <i>column/expression</i> maka akan terpotong sebanyak n karakter
<i>LTRIM(column/expression,['string2'])</i> <i>RTRIM(column/expression,['string2'])</i>	Menghapus karakter di bagian kiri (LTRIM) atau kanan (RTRIM) sehingga tidak diawali atau diakhiri dengan sembarang karakter pada <i>string2</i> . Default <i>string2</i> adalah karakter <i>special</i>
<i>INSTR(column/expression,'string2',[m][n])</i>	Mencari <i>string2</i> di dalam <i>column/expression</i> dengan awal pencarian adalah karakter ke m dan <i>string2</i> ke n dalam <i>column/expression</i> , default m dan n adalah 1
<i>REPLACE(text,search_string,[replacement_string])</i>	Mengganti semua <i>search_string</i> yang terdapat pada <i>text</i> dengan <i>replacement_string</i> . Jika <i>replacement_string</i> tidak ditulis, maka <i>search_string</i> pada <i>text</i> akan dihapus
<i>TRANSLATE(Str1,Str2,[Str3])</i>	Mengganti karakter penyusun <i>Str2</i> pada <i>Str1</i> dengan karakter penyusun <i>Str3</i>

<pre>❖ SQL> SELECT lower ('EduCatiOn') lower, UPPER ('EduCatiOn') upper, INITCAP ('EduCatiOn') initcap FROM dual;</pre>	<pre>❖ SQL> SELECT SUBSTR ('InstitutTeknologiTelkom',4) tes, SUBSTR ('InstitutTeknologiTelkom ',2,7) tas, SUBSTR ('InstitutTeknologiTelkom ', -6) tos FROM dual;</pre>																							
<pre>LOWER UPPER INITCAP ----- ----- ----- education EDUCATION Education</pre>	<pre>TES TAS TOS ----- ----- ----- titutTeknologiTelkom Institut elkom</pre>																							
Misal terdapat string 'how do you do' dengan posisi karakter sebagai berikut :																								
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>H</td><td>O</td><td>W</td><td>D</td><td>O</td><td>Y</td><td>O</td><td>U</td><td>D</td><td>O</td> </tr> <tr> <td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td> </tr> </table>	H	O	W	D	O	Y	O	U	D	O	1	2	3	4	5	6	7	8	9	10	11	12	13	
H	O	W	D	O	Y	O	U	D	O															
1	2	3	4	5	6	7	8	9	10	11	12	13												
<pre>❖ SQL> SELECT INSTR ('how do you do','do') ini1, INSTR ('how do you do ','do',4) ini2, INSTR ('how do you do ','do',1,2) ini3, INSTR ('how do you do ','do',-1) ini4 FROM dual;</pre>																								
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th>INI1</th><th>INI2</th><th>INI3</th><th>INI4</th> <th>LPAD1</th><th>R PAD</th><th>LPAD2</th> </tr> <tr> <td>5</td><td>5</td><td>12</td><td>12</td> <td>Knowledge</td><td>Knowledge</td><td>Knowl</td> </tr> </table>	INI1	INI2	INI3	INI4	LPAD1	R PAD	LPAD2	5	5	12	12	Knowledge	Knowledge	Knowl	<pre>❖ SQL> SELECT LPAD ('Knowledge',15,'.') lpad1, RPAD ('Knowledge', 15,'.') rpad, LPAD ('Knowledge', 5) lpad2 FROM dual;</pre>									
INI1	INI2	INI3	INI4	LPAD1	R PAD	LPAD2																		
5	5	12	12	Knowledge	Knowledge	Knowl																		
<pre>❖ SQL> SELECT LTRIM (nmSupervisor,'AFM') ltrim FROM supervisor; LTRIM -----</pre>																								
<pre>mita nto Wijaya Candra Ano arkhan Dani Zafira Annisa</pre>																								
<pre>❖ SQL> SELECT TRANSLATE ('Do Your Best Everytime And Everywhere','Aieuo','41t30') translate TRANSLATE -----</pre>																								
<pre>DO Y0tr B3st Ev3ryt1m3 And Ev3rygh3r3</pre>																								
<pre>❖ SQL> SELECT REPLACE ('I am here','am','am not'); Kalimat awal : I am here Kalimat setelah di replace : I am not her</pre>																								

Gambar 3-5. Contoh implementasi

2. Fungsi Tanggal dan Waktu

Masukkan tutorial cara mengubah lokalisasi bahasa di Oracle. Karena kesalahan format tanggal kemungkinan bisa dikarenakan ini.

Contoh: alter session set nls_sort = English;

Masukkan juga cara mengubah format tanggal

Contoh: alter session set nls_date_format='<my_format>';

Format tanggal:

Tabel 3-4. Format tanggal

MM	Numeric month (e.g., 07)
MON	Abbreviated month name (e.g., JUL)
MONTH	Full month name (e.g., JULY)
DD	Day of month (e.g., 24)
DY	Abbreviated name of day (e.g., FRI)
YYYY	4-digit year (e.g., 1998)
YY	Last 2 digits of the year (e.g., 98)
RR	Like YY, but the two digits are ``rounded'' to a year in the range 1950 to 2049. Thus, 06 is considered 2006 instead of 1906
AM (or PM)	Meridian indicator
HH	Hour of Day (1-12)
HH24	Hour of Day (0-23)
MI	Minute (0-59)
SS	Second (0-59)

ADD_MONTHS(d,n)	Menghasilkan sebuah tanggal yang berjarak <i>n</i> bulan setelah tanggal <i>d</i>
SYSDATE	Menghasilkan tanggal dan jam computer/system
LAST_DAY(d)	Menghasilkan tanggal terakhir dari bulan pada tanggal <i>d</i>
NEXT_DAY(d,day_name)	Menghasilkan tanggal yang jatuh pada hari <i>day_name</i> yang terdekat setelah tanggal <i>d</i>
MONTHS_BETWEEN(d1,d2)	Menghasilkan selisih bulan diantara dua tanggal
ROUND(d,[MONTH YEAR])	Menghasilkan tanggal pembulatan <i>d</i> berdasarkan MONTH atau YEAR
TRUNC(d,[MONTH YEAR])	Menghasilkan tanggal pemotongan <i>d</i> berdasarkan MONTH atau YEAR

```

❖ SQL> SELECT add_months ('14-SEP-2010',2) add_months FROM dual;
          ADD_MONTH
          -----
          14-NOV-10

❖ SQL> SELECT last_day ('24-MAR-2000') last_day FROM dual;
          LAST_DAY
          -----
          31-MAR-00

❖ SQL> SELECT next_day('10-JUL-2010','saturday') next_day FROM dual;
          NEXT_DAY
          -----
          17-JUL-10

❖ SQL> SELECT months_between('20-JUN-2011','01-JAN-2010') month_between FROM dual;
          MONTH_BETWEEN
          -----
          17.6129832

Asumsikan sysdate = '29-sep-2009'
❖ SQL> SELECT round(sysdate,'month') round1 FROM dual; →
          ROUND1
          -----
          01-OCT-09

❖ SQL> select round(sysdate,'year') round2 FROM DUAL; →
          ROUND2
          -----
          01-JAN-10

❖ SQL> SELECT trunc(sysdate,'month') trunc1 FROM dual; →
          TRUNC1
          -----
          01-SEP-09

❖ SQL> SELECT trunc(sysdate,'year') trunc2 FROM dual; →

```

Gambar 3-6. Implementasi tanggal

3. Fungsi Konversi

Tabel 3-5. Fungsi konversi

TO_NUMBER (char)	Mengkonversi tipe data varchar2, char ke Number
TO_CHAR (date)	Mengkonversi tipe data date ke varchar2, char
TO_CHAR (number)	Mengkonversi tipe data number ke varchar2
TO_DATE (char)	Mengkonversi tipe data char ke date

Implementasi fungsi konversi

SQL> SELECT to_number('8879') to_number FROM dual;
SQL> SELECT to_char('4-jul-2010') ch_to_date, to_char(817989212) ch_to_num FROM dual;

4. Fungsi Numerik

Tabel 3-6. Fungsi numerik

ABS(n)	Menghasilkan nilai absolut dari n
LN(n)	Menghasilkan nilai logaritma natural dari n , dimana $n>0$
LOG(m,n)	Menghasilkan nilai logaritma berbasis m untuk nilai n , dimana $m>1$ dan $n>0$
SIN(n)	Menghasilkan nilai sinus dari n
COS (n)	Menghasilkan nilai cosines dari n
TAN (n)	Menghasilkan nilai tangent dari n
MOD (m,n)	Menghasilkan sisa pembagian m oleh n , jika $n=0$ maka fungsi ini akan menghasilkan nilai m
SQRT (n)	Menghasilkan akar positif dari n , dimana tidak boleh negative
POWER (m,n)	Menghasilkan nilai m yang dikalikan sebanyak n kali
EXP (n)	Menghasilkan nilai e yang dikalikan sebanyak n kali, $3 = 2.71828183$
ROUND (n[,m])	Menghasilkan nilai n yang dibulatkan sebanyak m angka. Jika m tidak ditulis maka pembulatan sampai 0 angka
FLOOR (n)	Untuk pembulatan bilangan ke bawah
CEIL (n)	Untuk pembulatan bilangan ke atas
SIGN (n)	Menghasilkan nilai 1 jika $n>0$, nilai 0 jika $n=0$, dan nilai -1 jika $n<0$
TRUNC (n[,m])	Menghasilkan nilai n sampai m angka di belakang koma

Implementasi fungsi *numeric*

```
SQL> SELECT LOG(2,5) log, ABS(-8) abs, LN(2) ln FROM dual;
SQL> SELECT sin(30*3.14/180) sin30,
          cos(60*3.14/180) cos60,
          tan(45*3.14/180) tan45
     FROM dual
SQL> SELECT FLOOR(2.77) floor, CEIL(2.49) ceil FROM dual;
SQL> SELECT id, NVL(col1,'ZERO') AS output FROM ull_test_tab ORDER BY id;
SQL> SELECT id, NVL2(col1, col2, col3) AS output FROM null_test_tab ORDER BY id;
SQL> SELECT id, NULLIF(col3,col4) AS output FROM null_test_tab ORDER BY id;
SQL> SELECT id, COALESCE(col1, col2, col3) AS output FROM null_test_tab ORDER BY id;
```

5. Fungsi Umum

Tabel 3-7. Fungsi umum

NVL(ex1,ex2)	Menggantikan nilai dengan ex2 apabila nilai ditemukan pada ex1 adalah null
NVL2(ex1,ex2,ex3)	Menguji nilai ex1, kemudian memberikan nilai ex2 apabila ex1 NOT NULL dan memberikan nilai ex3 jika ex1 bernilai NULL
NULLIF(ex1,ex2)	Mengembalikan nilai NULL jika ex1 sama dengan ex2, dan akan mengembalikan nilai ex1 jika nilai ex1 tidak sama dengan ex2
COALESCE(ex1,ex2,...,exN)	Menghasilkan nilai ex pertama yang tidak NULL, fungsi ini akan menghasilkan nilai NULL jika semua ekspresi bernilai NULL

Implementasi fungsi umum

```
SQL> SELECT nmmember, alamat, nvl(telp, 'tidak punya nomor telepon') telepon FROM member;
```

```
SQL> SELECT nmmember, alamat, nvl2(telp,'punya telepon','tidak punya nomor telepon') telepon FROM memeber
```

3.2. Studi Kasus

- Buatlah sintaks untuk menampilkan semua teater tetapi harganya didiskon 15% dari harga awal.

Sebelum diskon 15% :

NOMOR_TEATER	KELAS	HARGA
1 Teater 1	Reguler	30000
2 Teater 2	Sweetbox	100000
3 Teater 3	4D	75000
4 Teater 4	Velvet	80000
5 Teater 5	3D	50000

Sesudah diskon 15%

NOMOR_TEATER	KELAS	Harga Diskon
1 Teater 1	Reguler	25500
2 Teater 2	Sweetbox	85000
3 Teater 3	4D	63750
4 Teater 4	Velvet	68000
5 Teater 5	3D	42500

- Buatlah satu *query* untuk menampilkan total pemasukan masing – masing teater

NOMOR_TE...	TOTAL
1 Teater 2	500000
2 Teater 1	150000

Modul 4. Scripting SQL – Pemrosesan Banyak Tabel

Tujuan Praktikum

1. Mahasiswa mampu memahami konsep dan jenis-jenis Join pada Basis Data Relasional
2. Mahasiswa mampu menerapkan jenis-jenis Join pada SQL
3. Mahasiswa mampu membuat *query* dari beberapa tabel dengan menggunakan jenis Join yang tepat

4.1. Ringkasan Materi

SQL juga menyediakan operasi yang dapat digunakan untuk memperoleh data hasil *Query* yang didapat dari dua tabel atau lebih. Operasi tersebut dikenal dengan istilah *join*. Penggabungan tabel-tabel tersebut harus melibatkan suatu kondisi yang dapat membentuk hubungan antara satu tabel dengan tabel lainnya. Kondisi penggabungan tersebut dikenal dengan istilah *join condition* (*Join* yang dipakai mengacu pada SQL 1999). Selain *Join* yang terdapat pada SQL 1999, pada Oracle juga terdapat 4 buah *join* yang digunakan pada versi 8i, yaitu *EquiJoin*, *Non-EquiJoin*, *Outer Join*, dan *Self Join*. *Join* pada SQL 1999 terdiri dari 3 jenis utama, yaitu : *Cross Join*, *Natural Join*, dan *Outer Join*.

4.1.1. Cross Join

Cross Join bisa diartikan sebagai *cross-product* pada dua tabel (*Cartesian Product*). *Cartesian product* adalah penggabungan dua tabel atau lebih tanpa *join condition* yang harus dipenuhi. Jumlah baris data yang dihasilkan adalah jumlah baris data tabel kiri **dikali** jumlah baris data tabel kanan.

Contoh selanjutnya digunakan tabel berikut :

Contoh:

```
SELECT idproduk, supervisor.nip, nmSupervisor  
FROM pengiriman  
CROSS JOIN supervisor;
```

Query di atas sama dengan:

```
SELECT idproduk, supervisor.nip, nmSupervisor  
FROM pengiriman, supervisor;
```

Untuk memudahkan dalam memahami hasil dari *CROSS JOIN* dapat dilihat contoh berikut :

Tabel 4-1. Tabel entitas A

A	
Nama	Keahlian
Adit	Java
Yusuf	C++
Geriska	Oracle

Tabel 4-2. Tabel entitas B

B	
NIP	No_Telp
11101	087821391145
11102	081220830014

Berikut ini *query select* yang menghasilkan hasil yang sama :

```
SQL> SELECT * FROM A CROSS JOIN B ;
```

```
SQL> SELECT * FROM A, B ;
```

Tabel hasil untuk statement di atas :

Tabel 4-3. Tabel hasil join

Nama	Keahlian	NIP	No_Telp
Adit	Java	11101	087821391145
Adit	Java	11102	081220830014
Yusuf	C++	11101	087821391145
Yusuf	C++	11102	081220830014
Geriska	Oracle	11101	087821391145
Geriska	Oracle	11102	081220830014

4.1.2. Natural Join

Sebuah **Natural Join** akan mengambil baris dari tabel-tabel yang mempunyai kesamaan pada semua nama kolom. Jika ada sebuah kolom mempunyai nama yang sama akan tetapi tipe datanya berbeda, maka Oracle akan mengembalikan *error message*.

Asumsi tabel member dan transaksi mempunyai satu kolom yang sama, yaitu kolom idmember.

```
SELECT idmember, nmMember, tgltransaksi  
FROM member  
NATURAL JOIN transaksi ;
```

Query di atas sama dengan query *EquiJoin* yang terdapat pada Oracle 8i:

```
SELECT idmember, nmMember, tgltransaksi  
FROM member, transaksi  
WHERE member.idmember = transaksi.idmember;
```

Sintaks query untuk *Natural Join*:

```
SELECT * FROM member  
NATURAL JOIN transaksi;
```

Maka tabel yang dihasilkan:

IDMEMB	NMMEMBER	TGLTRANSA
ME 0002	Deni Dermawan	03-APR-14
ME 0001	Anastasya AA	03-APR-14
ME 0002	Deni Dermawan	05-MAY-14
ME 0003	Muh Mukifatul	18-MAY-14
ME 0005	Fandi Arfandi	02-JUL-14

Gambar 4-1. Output

4.1.3. Join Using

Jika sebagian kolom mempunyai nama yang sama, akan tetapi semua atau ada beberapa tipe data yang tidak sama, maka pada *natural join* ini kita bisa saja memodifikasi menjadi sebuah *join* dengan menggunakan klausa **USING**. Kolom pada klausa *USING* haruslah tidak mempunyai sebuah *qualifier* atau alias pada SQL statement (tidak boleh ada prefix).

Contoh :

```
SELECT idmember, nmMember, tgltransaksi  
FROM member  
JOIN transaksi  
USING (idmember) ;
```

Query di atas sama dengan :

```
SELECT idmember, nmMember, tgltransaksi  
FROM member, transaksi  
WHERE member.idmember = transaksi.idmember;
```

4.1.4. Join On

Klausa **ON** digunakan untuk menspesifikasikan kolom yang digunakan pada *join*. Klausa ini juga dapat digunakan untuk menspesifikasikan sebuah *join* pada kolom yang mempunyai nama yang berbeda.

Contoh:

Contoh ini mempunyai menggunakan asumsi yang sama dengan contoh sebelumnya.

```
SELECT idmember, nmMember, tgltransaksi  
FROM member  
JOIN transaksi  
ON (member .idmember=transaksi. idmember) ;
```

Join pada table dirinya sendiri atau pada Oracle 8i disebut dengan *Self Join*. *studi kasus lain

```
SELECT p.nama_petugas, t.nama_manager  
FROM petugas p  
JOIN petugas t  
ON (p.id_petugas=t.id_petugas) ;
```

4.1.5. Outer Join

Join dari dua tabel yang mengembalikan hanya *row-row* yang sama jumlahnya pada setiap kolomnya disebut dengan **INNER JOIN**. Penggabungan dua buah *table* atau lebih yang menghasilkan baris-baris data di mana kolom-kolom yang menjadi *join condition* merupakan kolom yang harus memiliki nilai (tidak boleh null). Jika sebuah *join* antara dua buah tabel tersebut mengembalikan *row* yang di antara kolomnya tidak sama pada kiri atau kanan tabel, maka join tersebut disebut **LEFT / RIGHT OUTER JOIN**.

4.1.6. Left Outer Join

Join antara dua buah tabel yang mengembalikan hasil dengan baris boleh NULL pada **kiri** kolom tabel. Atau bisa dikatakan, *join* antara dua tabel yang menampilkan semua data di tabel kiri, walaupun data di tabel sebelah kanannya bernilai **NULL**.

Contoh:

```
-----  
SELECT *  
FROM pengiriman LEFT OUTER JOIN supervisor USING(nip);
```

Maka tabel yang dihasilkan:

NIP	IDPROD	IDDIST	NMSUPERVISOR	THN_MASUK
SUP001	000007	DIS004	Amita	2010
SUP001	000001	DIS001	Amita	2010
SUP002	000002	DIS001	Anto Wijaya	2010
SUP003	000005	DIS005	Candra Ano	2011
SUP003	000003	DIS002	Candra Ano	2011
SUP005	000006	DIS002	Zafira Annisa	2013
SUP005	000004	DIS003	Zafira Annisa	2013

7 rows selected.

Gambar 4-2. Output left outer join

4.1.7. Right Outer Join

Join antara dua buah tabel yang mengembalikan hasil dengan baris boleh *NULL* pada kanan kolom tabel. Atau bisa dikatakan, *join* antara dua tabel yang menampilkan semua data di tabel kanan, walaupun data di sebelah kirinya bernilai *NULL*.

Sintaks *query* untuk *RIGHT OUTER JOIN*,

Contoh:

```
SELECT *
FROM pengiriman RIGHT OUTER JOIN supervisor USING(nip);
```

Maka tabel yang dihasilkan:

NIP	IDPROD	IDDIST	NMSUPERVISOR	THN_MASUK
SUP001	000001	DIS001	Amita	2010
SUP002	000002	DIS001	Anto Wijaya	2010
SUP003	000003	DIS002	Candra Ano	2011
SUP005	000004	DIS003	Zafira Annisa	2013
SUP003	000005	DIS005	Candra Ano	2011
SUP005	000006	DIS002	Zafira Annisa	2013
SUP001	000007	DIS004	Amita	2010
SUP004			Farkhan Dani	2012

8 rows selected.

Gambar 4-3. Output right outer join

4.1.8. Full Outer Join

Join antara dua buah tabel yang mengembalikan hasil dengan baris boleh *NULL* pada bagian **KIRI** dan **KANAN** tabel.

Contoh:

```
SELECT *
FROM pengiriman FULL OUTER JOIN supervisor USING(nip);
```

Maka tabel yang dihasilkan:

NIP	IDPROD	IDDIST	NMSUPERVISOR	THN_MASUK
SUP001	000007	DIS004	Amita	2010
SUP001	000001	DIS001	Amita	2010
SUP002	000002	DIS001	Anto Wijaya	2010
SUP003	000005	DIS005	Candra Ano	2011
SUP003	000003	DIS002	Candra Ano	2011
SUP005	000006	DIS002	Zafira Annisa	2013
SUP005	000004	DIS003	Zafira Annisa	2013
SUP004			Farkhan Dani	2012

8 rows selected.

Gambar 4-4. Output full outer join

4.1.9. Non-Equijoin

Non equijoin terdapat dalam Oracle 8i atau SQL 9i. *Non-Equijoin* menghubungkan antara dua buah tabel yang tidak harus sama kolom pembandingnya.

Misalnya, kita akan menggabungkan tabel Mahasiswa dengan tabel Grade_Nilai seperti berikut ini:

Tabel 4-4. Tabel mahasiswa

Tabel Mahasiswa	
Nama	Nilai
Galuh	86

Dada	54
Mediani	44
Ananda	90

Tabel 4-5. Tabel grade nilai

Tabel Grade Nilai		
Grade	Batas_Bawah	Batas_Atas
A	81	100
B	61	80
C	41	60
D	21	40
E	0	20

Contoh:

```
SELECT m. Nama, m.Nilai, g.Grade  
FROM mahasiswa m, Grade_Nilai g  
WHERE m.Nilai BETWEEN g.Batas_Bawah AND g.Batas_Atas ;
```

Hasil *query* di atas adalah sebagai berikut:

Tabel 11-6. Hasil query

Nama	Nilai	Grade
Galuh	86	A
Dada	54	C
Mediani	44	C
Ananda	90	A

4.2. Contoh Kasus:

Pegawai ingin melihat seluruh data teater beserta seluruh kursi yang ada pada masing-masing teater. Informasi yang ingin ditampilkan adalah sebagai berikut:

```

NOMOR_TEATER NO_K
-----
Teater 1    A1
Teater 1    A2
Teater 1    A3
Teater 1    A4
Teater 1    A5
Teater 1    A6
Teater 1    A7
Teater 1    A8
Teater 1    B1
Teater 1    B2
              ...
Teater 5    E3
Teater 5    E4
Teater 5    E5
Teater 5    E6

```

NOMOR_TEATER NO_K

Teater 5 E7

Teater 5 E8

200 rows selected.

Gambar 4-5. Contoh kasus

Langkah-langkah pembuatan *query*:

1. Fokus pada tabel TEATER dan tabel KURSI. Jadikan tabel TEATER sebagai tabel utama.
 2. *JOIN* tabel TEATER dengan tabel KURSI pada atribut nomor_teater.
 3. Gunakan atribut nomor_teater dan nomor_kursi.
 4. Tuliskan dalam *syntax query!*

4.3. Studi Kasus

Studi kasus pada modul 7 ini menggunakan Skema Relasi pada modul 3 dan DDL-DML pada modul 5. Silahkan gunakan jenis *JOIN* yang mungkin digunakan (USING, JOIN ON, WHERE <kondisi>). Silahkan tambahkan kondisi yang perlu digunakan.

Digunakan himpunan entitas FILM, TEATER, KURSI, MEMBER beserta relasi yang menghubungkan himpunan entitas-himpunan entitas tersebut.

Anda ada seorang *programmer* yang memiliki akses ke *database* bioskop. Buatlah *query* untuk menampilkan *output* sesuai dengan yang diinginkan pada soal!

1. Konsumen ingin melihat film apa saja yang sedang di putar hari ini (asumsi hari ini adalah 5 Juli 2019). Informasi yang ingin dilihat konsumen tersebut adalah judul film, diputar di teater nomor berapa dan harga tiketnya.
(Hint : tambahkan kondisi hari ini (SYSDATE) berada dalam selang waktu atribut periode_start dan periode_end)

The screenshot shows the Oracle SQL Developer interface with a 'Query Result' tab open. The results show a list of movies with their titles, theater numbers, and prices. The data is as follows:

JUDUL	NOMOR_TEATER	HARGA
1 Keluarga Cemara	Teater 1	30000
2 Keluarga Cemara	Teater 1	30000
3 Keluarga Cemara	Teater 2	100000
4 Taufiq	Teater 2	100000
5 Buya Hamka	Teater 2	100000
6 Keluarga Cemara	Teater 1	30000
7 Habibie Ainun 3	Teater 1	30000
8 Habibie Ainun 3	Teater 2	100000
9 Habibie Ainun 3	Teater 2	100000
10 Habibie Ainun 3	Teater 1	30000

2. Konsumen ingin mengecek ketersediaan film Habibie Ainun 3. Dia ingin mengetahui akan diputar di teater nomor berapa, tanggal berapa, jam berapa.
(hint: gunakan operator = atau LIKE atau IN atau INSTR).

The screenshot shows the Oracle SQL Developer interface with a 'Query Result' tab open. The results show the screening schedule for the movie 'Habibie Ainun 3'. The data is as follows:

JUDUL	NOMOR_TEATER	PERIODE_START	PERIODE_END
1 Habibie Ainun 3	Teater 1	01-JUL-19	07-JUL-19
2 Habibie Ainun 3	Teater 2	01-JUL-19	07-JUL-19
3 Habibie Ainun 3	Teater 2	01-JUL-19	07-JUL-19
4 Habibie Ainun 3	Teater 1	01-JUL-19	07-JUL-19

3. Manajer ingin mengetahui member yang membeli tiket pada pemutaran film Keluarga Cemara pada Teater 1. Informasi yang ingin dilihatnya adalah id_member, tanggal beli dan status.

The screenshot shows the Oracle SQL Developer interface with a 'Query Result' tab open. The results show the purchase history for the movie 'Keluarga Cemara' at Teater 1. The data is as follows:

ID_MEMBER	TANGGAL	STATUS
1 MM0111	15-DEC-19	pesan
2 MM0114	18-DEC-19	pesan
3 MM0116	20-DEC-19	bayar
4 MM0118	22-DEC-19	pesan

4. Manajer sedang membuat pemetaan statistik tentang usia pengunjung bioskop beserta tempat duduk yang sering didudukinya. Dia ingin mengetahui member dengan umur lebih dari 32 tahun siapa saja sering duduk di kursi mana saja.

(Hint: Gunakan perhitungan $\text{ceil}((\text{sysdate}-\text{tgl_lahir})/365)$ untuk melakukan perhitungan umur)

The screenshot shows a SQL developer interface with two tabs: 'Script Output' and 'Query Result'. The 'Query Result' tab is active, displaying a table with four columns: ID_MEMBER, NAMA_MEMBER, UMUR, and NO_KURSI. The data is as follows:

ID_MEMBER	NAMA_MEMBER	UMUR	NO_KURSI
1 MM0114	Rahmi	38 A7	
2 MM0118	Tati	35 A3	
3 MM0112	Budi	35 B1	
4 MM0113	Ari	36 A8	
5 MM0115	Fahmi	34 A6	
6 MM0117	Doni	34 A2	

4.4. Tutorial

- Identifikasi data dan sumber data yang diperlukan
Jika terdapat lebih dari 1 sumber data (tabel), tentukan JOIN yang akan digunakan
- Tentukan operasi manipulasi data yang perlu digunakan untuk bisa menghasilkan data yang diinginkan
- Tambahkan kondisi jika diperlukan!
- Tuliskan *syntax query* berdasar langkah sebelumnya.
Jika terdapat operasi JOIN, perhatikan atribut *key* yang digunakan untuk JOIN Tabel!!
- Eksekusi *syntax query*
- Cek hasil *output query*
Jika *output* tidak sesuai, ulangi dari langkah 1
Jika terdapat *error*, ulangi dari langkah 3

Modul 5. Scripting Aggregate Function

Tujuan Praktikum

1. Mengidentifikasi *group function* yang tersedia
2. Menggunakan *group function* sesuai dengan tipe data yang akan diproses
3. Melakukan pengelompokan data dengan menggunakan GROUP BY
4. Memasukkan atau mengeluarkan data yang telah di kelompokan dengan menggunakan HAVING

5.1. Ringkasan Materi

Group function adalah suatu fungsi yang bekerja dengan menggunakan sekumpulan baris sebagai *input* dan menghasilkan sebuah hasil. *Group function* bisa juga disebut sebagai *multiple rows function*. *Group function* umum digunakan untuk melakukan analisa terhadap data.

EMPLOYEES

DEPARTMENT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
80	10500
80	11000
80	8600
	7000
10	4400

...

20 rows selected.

The maximum salary in the EMPLOYEES table.

MAX(SALARY)
24000

Gambar 5-1. Hasil grup function

5.1.1. Jenis-jenis Group function

- **AVG** : *Group function* untuk mencari nilai rata-rata dari suatu kolom.
- **COUNT** : *Group function* untuk menghitung kemunculan baris di suatu kolom
- **MAX** : *Group function* untuk mencari nilai terbesar di suatu kolom
- **MIN** : *Group function* untuk mencari nilai terkecil di suatu kolom
- **SUM** : *Group function* untuk mencari total nilai di suatu kolom
- **VARIANCE** : *Group function* untuk mencari *variance* di suatu kolom
- **STDDEV** : *Group function* untuk mencari nilai standar deviasi di suatu kolom

5.1.2. Syntax Group Function

```
SELECT [column,] group_function(column), ...
  FROM    table
  [WHERE   condition]
  [HAVING group_function_condition]
  [GROUP BY column]
  [ORDER BY column];
```

5.1.3. Group Function dan tipe data

Tabel 5-1. Grup function dan tipe data

Function	Tipe Data			
	String	Numeric	Date / Time	*
COUNT	✓	✓	✓	✓
MIN	✓	✓	✓	X
MAX	✓	✓	✓	X
AVG	X	✓	X	X
SUM	X	✓	X	X

5.1.4. DISTINCT pada group function

Digunakan untuk membuang nilai duplikat sebelum diproses oleh *group function*

Contoh query:

```
SELECT COUNT(DISTINCT department_id) FROM employees;
```

Output:

```
COUNT(DISTINCTDEPARTMENT_ID)
-----
11
```

5.1.5. Group function dan null values

Group function tidak akan memperhitungkan nilai null.

Contoh query:

```
SELECT AVG(commission_pct) FROM employees;
```

Output:

```
AVG(COMMISSION_PCT)
-----
.222857143
```

5.1.6. GROUP BY rules

GROUP BY digunakan untuk mengelompokkan data hasil agregasi.

EMPLOYEES

DEPARTMENT_ID	SALARY
10	4400
20	13000
20	6000
50	5800
50	3500
50	3100
50	2500
50	2600
60	9000
60	6000
60	4200
80	10500
80	8600
80	11000
90	24000
90	17000

4400

9500

3500

6400

■ ■ ■
20 rows selected.

The average salary
In EMPLOYEES table for each
department

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

Gambar 12-2. Group by rules

1. Jika terdapat *group function* di dalam sebuah *query*, maka SEMUA kolom yang tidak memiliki *group function* HARUS dimasukkan di dalam *GROUP BY*

```
SELECT department_id, AVG(salary)  
FROM employees  
GROUP BY department_id;
```

Output:

DEPARTMENT_ID	AVG(SALARY)
100	8601.33333
30	4150
20	7000
90	19333.3333
20	9500
70	10000
110	10154
50	3475.55556
80	80 8955.88235
40	6500
60	5760
10	4400

2. GROUP BY dapat digunakan walaupun tanpa group function

Query:

```
SELECT AVG(salary)
FROM employees
GROUP BY department_id;
```

Output:

AVG(SALARY)

8601.33333
4150
7000
19333.3333
9500
10000
10154
3475.55556
80 8955.88235
6500
5760
4400

Query:

```
SELECT department_id dept_id, job_id, SUM(salary)
FROM employees
GROUP BY department_id, job_id;
```

Output:

DEPT_ID	JOB_ID	SUM(SALARY)
110	AC_ACCOUNT	8300
90	AD_VP	34000
50	ST_CLERK	55700
80	SA REP	243500
50	ST_MAN	36400
80	SA_MAN	61000
110	AC_MGR	12008
90	AD_PRES	24000
60	IT_PROG	28800
100	FI_MGR	12008
30	PU_CLERK	13900
50	SH_CLERK	64300
20	MK_MAN	13000
100	FI_ACCOUNT	39600
30	SA REP	7000
70	PR REP	10000
30	PU_MAN	11000
10	AD_ASST	4400
20	MK REP	6000

40	HR_REP	6500
----	--------	------

Jika aturan ini dilanggar, akan muncul *error* sebagai berikut:

```
SELECT department_id, job_id, SUM(salary)
      *
```

ERROR at line 1:

ORA-00979: not a GROUP BY expression

3. GROUP BY tidak dapat menggunakan *column alias*

```
SELECT department_id dept_id, job_id, SUM(salary)
  FROM employees
 GROUP BY dept_id, job_id;
```

Error:

```
GROUP BY dept_id, job_id
      *
```

ERROR at line 3:

ORA-00904: "DEPT_ID": invalid identifier

4. GROUP BY dapat diletakkan sebelum maupun sesudah *HAVING*
5. GROUP BY harus diletakkan sebelum *ORDER BY*

5.1.7. HAVING rules

1. HAVING hanya dapat dipergunakan untuk kondisi yang menggunakan *group function*.

Query:

```
SELECT department_id, MAX(salary)
  FROM employees
 GROUP BY department_id
 HAVING MAX(salary) > 10000;
```

Output:

DEPARTEMEN_ID	MAX(SALARY)
100	12008
30	11000
90	24000
20	1300
110	12008
80	14000

Query:

```
SELECT job_id, SUM(salary) payroll
  FROM employees
 WHERE job_id NOT LIKE '%REP%'
 GROUP BY job_id
 HAVING SUM(salary) > 10000
 ORDER BY SUM(salary);
```

Output:

JOB_ID	PAYROLL
PU_MAN	11000
AC_MGR	12008
FI_MGR	12008
MK_MAN	13000
PU_CLERK	13900
AD_PRES	24000
IT_PROG	28800
AD_VP	34000
ST_MAN	36400
FI_ACCOUNT	39600
ST_CLERK	55700
SA_MAN	61000
SH_CLERK	64300

2. HAVING tidak bisa menggunakan column alias

Query:

```
SELECT job_id, SUM(salary) payroll
FROM employees
WHERE job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING payroll > 10000
ORDER BY SUM(salary);
```

Error:

```
HAVING payroll > 10000
      *SH
ERROR at line 5:
ORA-00904: "PAYROLL": invalid identifier
```

3. WHERE tidak bisa digunakan untuk group function

Query:

```
SELECT department_id, AVG(salary)
FROM employees
WHERE AVG(salary)>10000
GROUP BY department_id;
WHERE AVG(salary)>10000
```

Error:

```
WHERE AVG(salary)>10000
      *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

5.1.8. Nesting Group Function

Digunakan untuk mencari nilai agregasi dari sekumpulan sub agregasi

```
SELECT MAX(AVG(salary))
FROM employees
GROUP BY department_id;
```

MAX(AVG(SALARY))

19333.3333

Gambar 5-3. Nesting group function

Contoh di atas adalah *query* untuk mencari gaji tertinggi di dari gaji tertinggi di setiap departemen.

5.1.9. Contoh

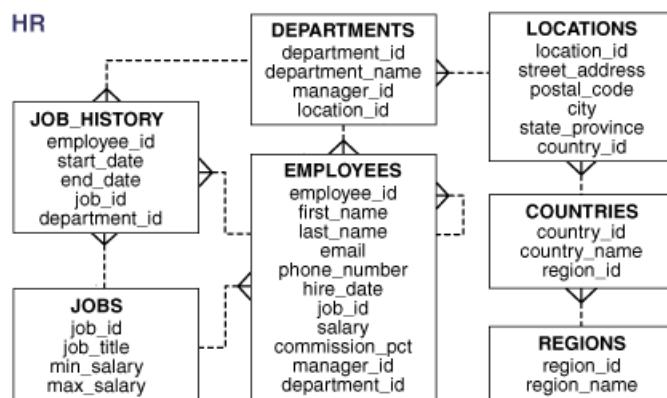
Studi Kasus

Untuk *section* contoh ini, Ide akan menggunakan contoh skema relasional “IDE” bawaan dari Oracle.

Skema ini default-nya di *lock*, sehingga untuk *unlock*-nya Ide harus *login* sebagai administrator dan menjalankan *script* sebagai berikut:

```
ALTER USER ide IDENTIFIED BY ide ACCOUNT UNLOCK;
```

Adapun skema relasionalnya sebagai berikut:



Gambar 12-4. Skema

Contoh 1: Cari tanggal rekrutasi pegawai yang terakhir direkrut.

Tanggal rekrutasi terakhir dari pegawai (pada table EMPLOYEES) dapat diketahui jika tanggal rekrutasi diurutkan dari yang pertama hingga terakhir, dan diambil yang terakhir. Terakhir disini bisa dianalogikan dengan MAX. Sehingga bentuk *query*-nya adalah

```
SELECT MAX(hire_date)
FROM employees;
```

Output-nya adalah

```
MAX(HIRE)
21-APR-08
```

Contoh 2: Cari masa kerja (dalam hari) pegawai terlama dan terbaru.

Sama dengan contoh kasus di atas, masa kerja terlama (direkrut pertama \ominus MIN) dan masa kerja terbaru (direkrut terakhir \ominus MAX) diambil dari table EMPLOYEES.

Masa kerja dapat dihitung dengan menggunakan sysdate (tanggal hari ini)

```
SELECT sysdate-MIN(hire_date), sysdate-MAX(hire_date)
FROM employees;
```

Output-nya:

SYSDATE-MIN(HIRE_DATE)	SYSDATE-MAX(HIRE_DATE)
-----	-----
5672.336	3017.3366

Contoh 3: Carilah kode department yang memiliki rata-rata gaji <\$5000 tanpa perlu menampilkan besarnya gaji tersebut.

Di kasus ini, yang harus diproses adalah kode *department* (*department_id*) dan rata-rata gaji (AVG(*salary*)). Karena rata-rata gaji tidak perlu ditampilkan, maka rata-rata gaji tidak perlu dimasukkan ke *SELECT*.

Karena terdapat kolom tanpa *group function* di *SELECT* (*department_id*), maka *query* ini HARUS menggunakan *GROUP BY*.

Karena ada kondisi pada *group function* dimana rata-rata gaji < \$5000, maka kondisi ini HARUS menggunakan *HAVING*.

Sehingga *query*-nya sebagai berikut:

```
SELECT department_id
FROM employees
HAVING AVG(salary)<5000
GROUP BY department_id;
```

Output:

DEPARTMENT_ID
30
50
10

5.2. Studi Kasus

Berdasarkan studi kasus pada modul 2, selesaikan permasalahan berikut dengan membuat perintah *query*-nya!

1. Cari berapa jumlah film yang tersedia di bioskop ini.
2. Cari berapa jumlah film yang sedang diputar dibioskop ini. Satu judul film hanya boleh dihitung satu kali.
3. Cari *id_member* yang sudah menonton lebih dari 2 kali. Lakukan hal yang sama untuk 3 kali, 4 kali dan 5 kali.

4. Cari member termuda dan jumlah film yang sudah ditontonnya.
5. Cari jumlah member yang lahir untuk setiap bulannya.
6. Berapa jumlah jadwal tayang per film pada periode aktif?

5.3. Tutorial

1. Identifikasi data dan sumber data yang diperlukan
Jika terdapat lebih dari 1 sumber data (tabel), tentukan *JOIN* yang akan digunakan
2. Tentukan operasi manipulasi data yang perlu digunakan untuk bisa menghasilkan data yang diinginkan
3. Tambahkan kondisi jika diperlukan!
Jika terdapat operasi *JOIN*, perhatikan atribut *key* yang digunakan untuk *JOIN* Tabel!
4. Tuliskan *syntax query* berdasar langkah sebelumnya.
Jika terdapat operasi *JOIN*, perhatikan atribut *key* yang digunakan untuk *JOIN* Tabel!
5. Eksekusi *syntax query*
6. Cek hasil *output query*
Jika *output* tidak sesuai, ulangi dari langkah 1
Jika terdapat *error*, ulangi dari langkah 3

Modul 6. Scripting Subquery & Nested Subquery

Tujuan Praktikum:

1. Mengidentifikasi masalah yang dapat diselesaikan dengan *subquery*
2. Memahami *subquery*
3. Membedakan antara *single-row* dan *multiple-row subquery*
4. Menggunakan *subquery* dalam DDL
5. Menggunakan *subquery* dalam DML
6. Menggunakan *subquery* dalam *in-line-view*
7. Menggunakan *correlated subquery*

6.1. Ringkasan Materi

6.1.1. Masalah Subquery

Subquery dipergunakan untuk menyelesaikan *query* yang nilainya bisa didapat dengan memanfaatkan *query* yang lain.

Contoh:

Siapakah yang memiliki gaji di atas \$11000? *Query* ini dapat langsung dijawab karena nilai gaji sudah terdefinisi langsung.

Siapakah yang memiliki gaji lebih besar dari pegawai bernama ‘Abel’? *Query* ini tidak bisa dijawab tanpa terlebih dulu mengetahui gaji Abel. Sehingga, masalah ini hanya bisa diselesaikan dengan *subquery*.

Dengan kata lain, *subquery* adalah *query* yang digunakan didalam *query*.

6.1.2. Syntax Subquery

```
SELECT [column], [column], ...
      FROM      table
      [WHERE condition] =
      (SELECT [column]
      FROM table
      [WHERE condition] )
```

6.1.3. Basic Subquery rules

1. *Subquery* harus berada didalam tanda kurung ().
2. *Subquery* HARUS berada disebelah KANAN operator.
3. *Subquery* tidak memerlukan ORDER BY, kecuali untuk Top-N Analysis.
4. *Single-row* operator HANYA untuk *single-row subquery*, dan *multiple-row* operator hanya untuk *multiple-row subquery*.

6.1.4. Subquery types: Single Row & Multiple Rows

1. *Single-row subquery* adalah *subquery* yang menghasilkan TEPAT satu baris *result*.
2. *Multiple-row subquery* adalah *subquery* yang menhasilkan LEBIH DARI satu baris *result*.

6.1.5. Single Row Subquery

6.1.5.1. Operator

Tabel 6-1. Operator

Operator	Arti
=	Sama dengan
>	Lebih besar dari
>=	Lebih besar dari atau sama dengan
<	Lebih kecil dari
<=	Lebih kecil dari atau sama dengan
<>	Tidak sama dengan

Contoh:

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id =
(SELECT job_id
FROM employees
WHERE employee_id = 141)
AND salary >
(SELECT salary
FROM employees
WHERE employee_id = 143);
```

Output:

LAST_NAME	JOB_ID	SALARY
Nayer	ST_CLERK	3200
Mikkilineni	ST_CLERK	2700
Bissot	ST_CLERK	3300
Atkinson	ST_CLERK	2800
Mallin	ST_CLERK	3300
Rogers	ST_CLERK	2900
Ladwig	ST_CLERK	3600
Stiles	ST_CLERK	3200
Seo	ST_CLERK	2700
Rajs	ST_CLERK	3500
Davies	ST_CLERK	3100

Jika dieksekusi terpisah, masing-masing subquery akan menghasilkan:

```
SELECT job_id
FROM employees
WHERE employee_id = 141;
```

Output:

```
JOB_ID
-----
ST_CLERK
```

Query:

```
SELECT salary  
FROM employees  
WHERE employee_id = 143;
```

Output:

```
SALARY  
-----  
2600
```

Sehingga, jika tanpa *subquery*, query di atas akan menjadi:

```
SELECT last_name, job_id, salary  
FROM employees  
WHERE job_id = 'ST_CLERK'  
AND salary > 2600;
```

Output:

LAST_NAME	JOB_ID	SALARY
Nayer	ST_CLERK	3200
Mikkilineni	ST_CLERK	2700
Bissot	ST_CLERK	3300
Atkinson	ST_CLERK	2800
Mallin	ST_CLERK	3300
Rogers	ST_CLERK	2900
Ladwig	ST_CLERK	3600
Stiles	ST_CLERK	3200
Seo	ST_CLERK	2700
Rajs	ST_CLERK	3500
Davies	ST_CLERK	3100

6.1.5.2. GROUP BY pada Subquery

Group by dapat dipergunakan pada *subquery* untuk menghasilkan satu nilai agregasi.

```
SELECT last_name, job_id, salary  
FROM employees  
WHERE salary =  
(SELECT MIN(salary)  
FROM employees);
```

Output:

LAST_NAME	JOB_ID	SALARY
Olson	ST_CLERK	2100

Hasil dari *subquery* tersebut adalah $\text{MIN}(\text{salary}) = 2100$.

Mengapa hasil dari agregasi *subquery* tidak memakai *HAVING*? Karena kondisi di *query* utama TIDAK menggunakan *grouping function*.

WHERE salary = ...

6.1.5.3. HAVING pada Subquery

Having dipergunakan jika ingin membatasi hasil dari operasi *group function*.

```
SELECT department_id, MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) >
(SELECT MIN(salary)
FROM employees
WHERE department_id = 50);
```

Output:

DEPARTMENT_ID	MIN(SALARY)
100	6900
30	2500
	7000
90	17000
20	6000
70	10000
110	8300
80	6100
40	6500
60	4200
10	4400

100	6900
30	2500
	7000
90	17000
20	6000
70	10000
110	8300
80	6100
40	6500
60	4200
10	4400

6.1.6. Multiple Row subquery

Multiple-row subquery adalah subquery yang akan menghasilkan lebih dari satu baris. Contoh berikut adalah query untuk gaji terendah per departemen.

```
SELECT MIN(salary)
FROM employees
GROUP BY department_id;
```

Output:

MIN(SALARY)
6900
2500
7000
17000
6000
10000
8300
2100
6100
6500
4200
4400

| 6900 |
| 2500 |
| 7000 |
| 17000 |
| 6000 |
| 10000 |
| 8300 |
| 2100 |
| 6100 |
| 6500 |
| 4200 |
| 4400 |

Jika *subquery* diproses menggunakan operator *single-row subquery* untuk *query* di atas, maka akan menghasilkan *error* sebagai berikut:

```
SELECT employee_id, last_name  
FROM employees  
WHERE salary =  
(SELECT MIN(salary)  
FROM employees  
GROUP BY department_id);
```

Error:

```
(SELECT MIN(salary)  
*  
ERROR at line 4:  
ORA-01427: single-row subquery returns more than one row
```

Untuk mengatasi *error* tersebut, operator *multiple-row subquery* harus dipergunakan.

6.1.6.1. IN operator

In operator digunakan untuk mencari baris data yang sesuai dengan salah satu *result* dari *subquery*.

Contoh berikut adalah untuk mencari pegawai yang bekerja sebagai posisi tertentu dimana posisi tersebut $\text{bergaji} > 15000$.

```
SELECT last_name, department_id  
FROM employees  
WHERE job_id IN (SELECT job_id  
FROM employees  
WHERE salary > 15000);
```

Output:

LAST_NAME	DEPARTMENT_ID
King	90
De Haan	90
Kochhar	90

6.1.6.2. ANY operator

ANY operator HARUS digabung penggunaannya dengan *single-row* operator, sehingga bentuknya menjadi:

Tabel 6-2. Operator ANY

Operator	Arti
=ANY	Sama dengan salah satu nilai dari hasil <i>subquery</i>
>ANY	Lebih besar dari nilai terkecil dari hasil <i>subquery</i>
>=ANY	Lebih besar atau sama dengan nilai terkecil dari hasil <i>subquery</i>
<ANY	Lebih kecil dari nilai terbesar dari hasil <i>subquery</i>
<=ANY	Lebih kecil atau sama dengan nilai terbesar dari hasil <i>subquery</i>
<>ANY	Tidak sama dengan salah satu hasil dari hasil <i>subquery</i>

Contoh: Mencari karyawan yang bergaji lebih kecil dari IT Programmer

Hasil Subquery:

```
SELECT salary  
FROM employees  
WHERE job_id = 'IT_PROG';
```

Output:

SALARY

9000
6000
4800
4800
4200

Query:

```
SELECT employee_id, last_name, job_id, salary  
FROM employees  
WHERE salary < ANY  
(SELECT salary  
FROM employees  
WHERE job_id = 'IT_PROG')  
AND job_id >< 'IT_PROG';
```

Output:

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
132	Olson	ST_CLERK	2100
136	Philtanker	ST_CLERK	2200
128	Markle	ST_CLERK	2200
135	Gee	ST_CLERK	2400
127	Landry	ST_CLERK	2400
191	Perkins	SH_CLERK	2500
182	Sullivan	SH_CLERK	2500
144	Vargas	ST_CLERK	2500
140	Patel	ST_CLERK	2500
131	Marlow	ST_CLERK	2500
119	Colmenares	PU_CLERK	2500
	...		
206	Gietz	AC_ACCOUNT	8300
177	Ivingston	SA REP	8400
	Taylor	SA REP	8600
175	Tton	SA REP	8800

6.1.6.3. ALL operator

ALL operator HARUS digabung dengan single-row operator.

Tabel 13-3. Operator ALL

Operator	Arti
=ALL	Sama dengan seluruh nilai dari hasil subquery
>ALL	Lebih besar dari nilai terbesar dari hasil subquery

Operator	Arti
>=ALL	Lebih besar atau sama dengan nilai terbesar dari hasil subquery
<ALL	Lebih kecil dari nilai terkecil dari hasil subquery
<=ALL	Lebih kecil atau sama dengan nilai terkecil dari hasil subquery
<>ALL	Tidak sama satupun dari hasil subquery

Contoh: Mencari karyawan yang bergaji lebih kecil dari seluruh IT Programmer.

Query:

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ALL
(SELECT salary
FROM employees
WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';
```

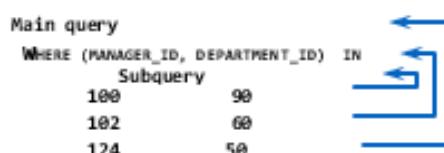
Output:

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
185	Bull	SH_CLERK	4100
192	Bell	SH_CLERK	4000
193	Everett	SH_CLERK	3900
188	Chung	SH_CLERK	3800
137	Ladwig	ST_CLERK	3600
189	Dilly	SH_CLERK	3600
141	Rajs	ST_CLERK	3500
186	Dellinger	SH_CLERK	3400
133	Mallin	ST_CLERK	3300
129	Bissot	ST_CLERK	3300
180	Taylor	SH_CLERK	3200
...			
128	Markle	ST_CLERK	2200
136	Philtanker	ST_CLERK	2200
132	son	ST_CLERK	2100

6.1.7. Multiple Column Subquery

Multiple column subquery dipergunakan jika ada PASANGAN nilai yang harus dipenuhi untuk *subquery* tersebut. *Multiple column subquery* HANYA BISA menggunakan *multiple-row operator*.

Contoh: Mencari pegawai yang bekerja dengan kombinasi manager dan departemen yang sama dengan ID pegawai 178 & 174.



Gambar 6-1. Multiple column subquery

Query:

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE (manager_id, department_id) IN
(SELECT manager_id, department_id
FROM employees
WHERE employee_id IN (178,174))
AND employee_id NOT IN (178,174);
```

Output:

EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
175	149	80
176	149	80
177	149	80
179	149	80

6.1.8. Subquery pada DDL

Subquery dapat dipergunakan untuk membuat *table* berdasarkan *query*. Tipe data akan mengikuti tipe data pada *table* asal, atau sesuai dengan data yang berada pada kolom tersebut.

```
CREATE TABLE employee_salary
AS
SELECT last_name, department_id, salary
FROM employees;
```

6.1.9. Subquery pada DML

6.1.9.1. INSERT

Insert dengan menggunakan *subquery* hanya dapat dilakukan jika *subquery* yang dihasilkan SAMA PERSIS dengan struktur *table* yang akan di-*INSERT*.

Contoh, misalnya terdapat *table* *employees_backup* dengan struktur yang sama persis dengan *employees*. Untuk memasukkan data dari *employees* ke *employees_backup* dapat dilakukan dengan cara:

```
INSERT INTO employees_backup
SELECT * FROM employees;
```

SQL> INSERT INTO employees_backup

1 SELECT * FROM employees;

6.1.9.2. UPDATE

Subquery pada *update* dapat diletakkan pada SET ataupun WHERE.

Contoh: *UPDATE* data karyawan 114 sesuai dengan karyawan 205.

```
UPDATE employees
SET job_id = (SELECT job_id
               FROM employees
              WHERE employee_id = 205),
    salary = (SELECT salary
              FROM employees
```

```

        WHERE employee_id = 205)
WHERE employee_id = 114;

```

Contoh berikut adalah mengganti departemen_id menjadi departemen_id yang sama dengan pegawai 100 untuk semua karyawan yang memiliki pekerjaan yang sama dengan pegawai 200.

```

UPDATE copy_emp
SET department_id = (SELECT department_id
                      FROM employees
                      WHERE employee_id = 100)
WHERE job_id = (SELECT job_id
                 FROM employees
                 WHERE employee_id = 200);

```

6.1.9.3. DELETE

Subquery pada *delete* HANYA dapat dipergunakan di *WHERE statement*.

Contoh: menghapus semua karyawan di departemen yang memiliki kata Public dan nama departemen berada di *table* terpisah.

```

DELETE FROM employees
WHERE department_id =
(SELECT department_id
FROM departments
WHERE department_name LIKE '%Public%');

```

6.1.10. In-Line-View Subquery

In-Line View adalah *subquery* yang diletakkan di *FROM clause*. *In-line-view* dapat diproses dengan *JOIN*, agregasi ataupun *query* yang lainnya.

Contoh berikut adalah mencari karyawan yang bekerja di suatu departemen dimana gaji karyawan tersebut lebih besar dari rata-rata gaji tempat dia bekerja.

Subquery / In-line-view disini dipakai untuk mengidentifikasi gaji rata-rata per departemen, untuk kemudian digabungkan dengan *table* karyawan dan dipilih hanya karyawan dengan gaji yang lebih besar dari gaji rata-rata.

```

SELECT a.last_name, a.salary, a.department_id, b.salavg
FROM employees a, (SELECT department_id,
AVG(salary) salavg
FROM employees
GROUP BY department_id) b
WHERE a.department_id = b.department_id
AND a.salary > b.salavg;

```

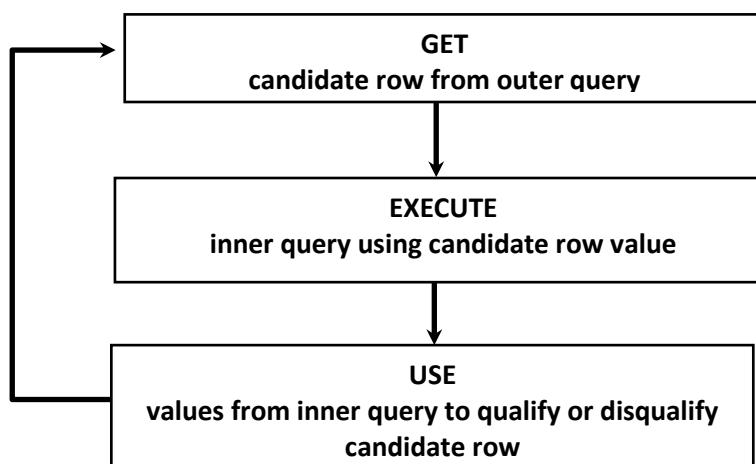
Output:

LAST_NAME	SALARY	DEPARTMENT_ID	SALAVG
King	24000	90	19333.3333
Hunold	9000	60	5760
Ernst	6000	60	5760
Greenberg	12008	100	8601.33333
Faviet	9000	100	8601.33333

Raphaely	11000	30	4150
Weiss	8000	50	3475.55556
Fripp	8200	50	3475.55556
Kaufling	7900	50	3475.55556
Vollman	6500	50	3475.55556
Mourgos	5800	50	3475.55556
...			
Everett	3900	50	3475.55556
Hartstein	13000	20	9500
Higgins	12008	110	10154

6.1.11. Correlated Subquery

Correlated subquery dipergunakan untuk eksekusi baris-per-baris. Setiap *subquery* diproses sebanyak jumlah baris di *outer query*. Dari sisi penggunaan, *correlated subquery* adalah bentuk *subquery* yang paling tidak efisien.



Gambar 6-2. Correlated subquery

6.1.11.1. Syntax

```

SELECT column1, column2, ...
FROM   table1
WHERE  column1 operator
       (SELECT column1, column2
        FROM   table2
        WHERE  expr1 = .expr2);
  
```

Gambar 6-3. Syntax

6.1.11.2. Correlated Query

Tampilkan karyawan yang memiliki gaji lebih besar dari rata-rata gaji di departemen tempat dia bekerja.

```

SELECT last_name, salary, department_id
FROM   employees outer
WHERE  salary >(SELECT AVG(salary)
                FROM   employees
                WHERE  department_id = outer.department_id);
  
```

Output:

LAST_NAME	SALARY	DEPARTMENT_ID
King	24000	90
Hunold	9000	60
Ernst	6000	60
Greenberg	12008	100
Faviet	9000	100
Raphaely	11000	30
Weiss	8000	50
Fripp	8200	50
Kaufling	7900	50
Vollman	6500	50
Mourgos	5800	50
...		
Everett	3900	50
Hartstein	13000	20
Higgins	12008	110

Tampilkan karyawan yang telah ganti pekerjaan minimal 2 x

```
SELECT e.employee_id, last_name,e.job_id
FROM employees e
WHERE 2 <= (SELECT COUNT(*)
FROM job_history
WHERE employee_id = e.employee_id);
```

Output:

EMPLOYEE_ID	LAST_NAME	JOB_ID
101	Kochhar	AD_VP
176	Taylor	SA_REP
200	alen	AD_ASST

6.1.11.3. Correlated DML

1. Correlated UPDATE

Mengubah data berdasarkan informasi dari data yang sedang diproses.

```
UPDATE table1 alias1
SET column = (SELECT expression
              FROM table2 alias2
              WHERE alias1.column =
                    alias2.column);
```

Gambar 6-4. Correlated DML

Contoh: Mendenormalisasi *table employees* dengan cara menambahkan kolom *department_name*, dan mengisi kolom tersebut sesuai *department_id* per baris.

```
ALTER TABLE employees
ADD(department_name VARCHAR2(14));
```

```

UPDATE employees e
SET department_name =
    (SELECT department_name
     FROM departments d
     WHERE e.department_id = d.department_id);

```

2. Correlated DELETE

```

UPDATE table1 alias1
SET column = (SELECT expression
               FROM table2 alias2
               WHERE alias1.column =
                     alias2.column);

```

Gambar 13-5. Correlated Delete

Contoh: Menghapus baris di employees jika data pegawai tersebut ada di table emp_history

```

DELETE FROM employees E
WHERE employee_id =
    (SELECT employee_id
     FROM emp_history
     WHERE employee_id = E.employee_id);

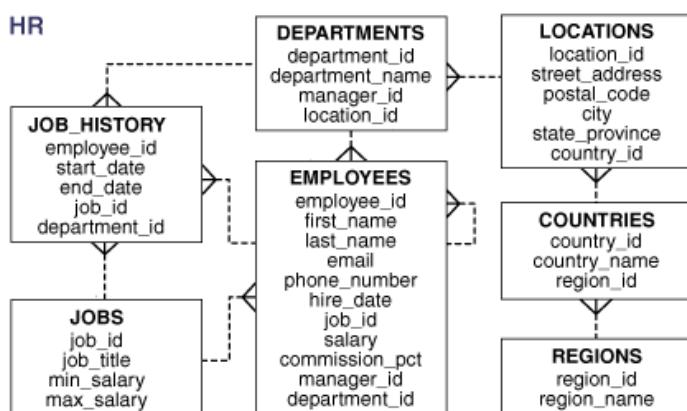
```

6.2. Contoh

Untuk section Contoh ini, Anda akan menggunakan contoh skema relasional “HR” bawaan dari Oracle. Skema ini default-nya di *lock*, sehingga untuk *unlock*-nya Anda harus *login* sebagai administrator dan menjalankan *script* sebagai berikut:

```
ALTER USER hr IDENTIFIED BY hr ACCOUNT UNLOCK;
```

Adapun skema relasionalnya sebagai berikut:



Gambar 13-6. Skema

Contoh 1: Cari nama(depan dan belakang) pegawai yang bekerja di department “IT”

Nama pegawai hanya dapat dilihat di table *EMPLOYEES*, sedangkan nama department hanya dapat dilihat di table *DEPARTMENTS*, sehingga salah satu bentuk *query* yang dapat dipergunakan adalah:

```
SELECT first_name||' '|last_name  
FROM employees  
WHERE department_id = (SELECT department_id  
FROM departments  
WHERE department_name='IT');
```

Output:

```
FIRST_NAME||"||LAST_NAME
```

```
-----  
Alexander Hunold  
Bruce Ernst  
David Austin  
Valli Pataballa  
Diana Lorentz
```

Contoh 2:

Carilah nama pegawai dengan masa kerja terlama.

Masa kerja terlama berarti pegawai yang direkrut pertama kali. Nilai ini dapat diperoleh dengan menggunakan MIN(hire_date). Setelah nilai ini didapat, nilai tersebut dapat digunakan di-query selanjutnya untuk mencari nama pegawai.

```
SELECT first_name||' '|last_name  
FROM employees  
WHERE hire_date = (SELECT MIN(hire_date)  
FROM employees);
```

Output:

```
FIRST_NAME||"||LAST_NAME
```

```
-----  
Lex De Haan
```

Contoh 3: Carilah nama pegawai dengan gaji dibawah gaji rata-rata perusahaan. Tampilkan nama department tempat mereka bekerja.

Nama department hanya bisa diambil dari table DEPARTMENT, sedangkan informasi gaji hanya dapat diambil dari table EMPLOYEES. Gaji rata-rata dapat diambil dengan menggunakan *subquery*, dan pegawai dapat difilter menggunakan hasil dari gaji rata-rata tersebut.

```
SELECT last_name, department_name  
FROM employees  
JOIN departments  
USING (department_id)  
where salary < ( SELECT AVG(salary)  
FROM employees)  
GROUP BY last_name, department_name;
```

Output:

LAST_NAME	DEPARTMENT_NAME
Colmenares	Purchasing
Himuro	Purchasing
Grant	Shipping

Landry	Shipping
Bissot	Shipping
Olson	Shipping
Philtanker	Shipping
Fleaur	Shipping
Sarchand	Shipping
Dellinger	Shipping
Bell	Shipping
...	
Rogers	Shipping
Geoni	Shipping
Dilly	Shipping

Contoh 4: Dari contoh 3, tampilkan juga rata-rata gaji diperusahaan tersebut.

Rata-rata gaji dapat diperoleh dengan menggunakan `AVG(salary)`, tetapi menggunakan fungsi ini langsung didalam `SELECT` akan mengakibatkan `AVG(salary)` dikelompokkan berdasarkan nama pegawai dan nama `department`, hal ini akan mengakibatkan kesalahan pengelompokan. Salah satu cara yang dapat dilakukan adalah dengan menggunakan *in-line-view* pada `SELECT`. Namun harus diperhatikan bahwa `sub-query` ini HARUS TEPAT menghasilkan satu baris/kolom (*scalar*), sehingga query-nya akan berupa:

```
SELECT last_name, department_name, (SELECT AVG(salary) FROM employees) avg_sal
FROM employees
JOIN departments
USING (department_id)
where salary < ( SELECT AVG(salary)
FROM employees)
GROUP BY last_name, department_name;
```

Output:

LAST_NAME	DEPARTMENT_NAME	AVG_SAL
Colmenares	Purchasing	6461.83178
Himuro	Purchasing	6461.83178
Grant	Shipping	6461.83178
Landry	Shipping	6461.83178
Bissot	Shipping	6461.83178
Olson	Shipping	6461.83178
Philtanker	Shipping	6461.83178
Fleaur	Shipping	6461.83178
Sarchand	Shipping	6461.83178
Dellinger	Shipping	6461.83178
Bell	Shipping	6461.83178
...		
Rogers	Shipping	6461.83178
Geoni	Shipping	6461.83178
Dilly	Shipping	6461.83178

Contoh 5: Dari contoh 3, tampilkan gaji rata-rata di department tempat mereka bekerja

Untuk kasus ini, ada beberapa modifikasi:

- *scalar in-line-view* harus diganti menjadi *scalar correlated subquery*.
- *GROUP BY* pada *outer query* dihilangkan, karena *scalar in-line-view* BUKAN *group function*
- *JOIN USING* tidak dapat dipergunakan dan diganti menjadi *JOIN ON*, karena *correlated subquery* membutuhkan *column identifier*.

Sehingga, bentuk *query*-nya menjadi:

```
SELECT last_name, department_name, (SELECT AVG(salary)
FROM employees
WHERE department_id = e.department_id) dept_avg_sal
FROM employees e
JOIN departments d
ON (e.department_id = d.department_id)
where salary < ( SELECT AVG(salary)
FROM employees);
```

Output:

LAST_NAME	DEPARTMENT_NAME	DEPT_AVG_SAL
Whalen	Administration	4400
Fay	Marketing	9500
Colmenares	Purchasing	4150
Himuro	Purchasing	4150
Khoo	Purchasing	4150
Baida	Purchasing	4150
Tobias	Purchasing	4150
Davies	Shipping	3475.55556
Grant	Shipping	3475.55556
OConnell	Shipping	3475.55556
Feeney	Shipping	3475.55556
...		
Banda	Sales	8955.88235
Ande	Sales	8955.88235
Johnson	Sales	8955.88235

6.3. Studi Kasus

Berdasarkan studi kasus pada modul 1 dan 2, selesaikan permasalahan berikut dengan membuat perintah *query* nya!

1. Tampilkan nama member dan umurnya, yang memesan tiket dengan film keluaran terlama
2. Tampilkan id_film, judul film dan banyak pemutaran dari film yang ditayangkan paling sering pada bioskop tersebut.
3. Tampilkan nomor_teater dan total_pemakaian teater tersebut yang paling jarang digunakan.

Script Output		Query Result	
SQL	All Rows Fetched		
NOMOR_TEATER	TOTAL		
1	Teater 3	1	

4. Buatlah sebuah table dengan nama history_order yang berisi id_order, id_member, nomor_teater dari table orderfilm dan kursi dengan ketentuan bahwa banyaknya nomor teater yang telah dipesan yaitu 6.
5. Insert ke table history_order dengan nilai yang didapat dari table orderfilm dan teater dengan ketentuan, id_member yang telah memesan merupakan id_member dengan umur tertua.

Script Output			Query Result		
SQL	All Rows Fetched:	1 in 0.004 second	ID_ORDER	ID_MEMBER	NOMOR_TEATER
			1	P10004	MM0114 Teater 2

6.4. Tutorial

1. Identifikasi data dan sumber data yang diperlukan
Jika terdapat lebih dari 1 sumber data (tabel), tentukan JOIN yang akan digunakan
2. Tentukan operasi manipulasi data yang perlu digunakan untuk bisa menghasilkan data yang diinginkan
3. Tambahkan kondisi jika diperlukan!
4. Tuliskan *syntax query* berdasar langkah sebelumnya.
Jika terdapat operasi JOIN, perhatikan atribut key yang digunakan untuk JOIN Tabel!
5. Eksekusi *syntax query*
6. Cek hasil *output query*
Jika *output* tidak sesuai, ulangi dari langkah 1
Jika terdapat *error*, ulangi dari langkah 3

Modul 7. Normalisasi

Tujuan Praktikum

- Praktikan mampu melakukan normalisasi data

7.1. Ringkasan Materi

Normalisasi adalah proses pengelolaan data dalam *database*. Termasuk didalamnya membuat tabel dan membangun hubungan antara tabel berdasarkan aturan perancangan yang baik agar *database* menjadi lebih fleksibel dan melindungi data dari faktor redundansi dan ketidakkonsistensi *dependencies* (ketergantungan fungsional). Normalisasi adalah proses formal untuk memutuskan atribut mana saja yang harus dikelompokkan bersama dalam satu relasi.

Normalisasi merupakan alat bantu utama untuk memvalidasi dan meningkatkan kualitas desain logis sehingga mampu memenuhi *constraint/batasan* yang ada dan menghindari duplikasi data yang tidak perlu. Teori normalisasi didasarkan pada konsep *normal form*. Normalisasi harus menghapus redundansi tetapi jangan sampai mengorbankan integritas data. Secara umum, proses normalisasi akan menghasilkan banyak entitas sederhana dari beberapa entitas semantik kompleks (tabel universal).

7.1.1. Tujuan Normalisasi

Normalisasi memungkinkan untuk menghilangkan anomali dan membantu menjaga konsistensi data dalam *database*.

- Untuk menghindari redundansi dengan menyimpan setiap fakta dalam *database* hanya sekali
- Untuk memasukkan data ke dalam bentuk yang lebih mampu mengakomodasi perubahan secara akurat
- Untuk menghindari memperbarui (*update*) data yang dapat menyebabkan anomali
- Untuk memfasilitas penegakan batasan-batasan (*constraint*) data
- Untuk menghindari *coding* yang tidak perlu. Penambahan dalam *trigger* dan *stored procedure* dapat diminta untuk menangani data yang tidak normal, dan hal tersebut dapat mempengaruhi kinerja *database* secara signifikan.

7.1.2. Tahapan dalam Normalisasi

Sebuah tabel relasional dikatakan memenuhi bentuk normal tertentu jika memenuhi satu set kondisi tertentu. Derajat normalisasi ditentukan menggunakan *normal form*. Secara umum ada 6 bentuk normal, namun yang akan dibahas pada bahasan ini hanya 4 dari 6, yaitu normal bentuk 1 (1 NF), normal bentuk 2 (2 NF), normal bentuk 3 (3 NF), dan terakhir adalah BCNF (*Boyce-Codd Normal Form*). Setiap normal form merupakan suatu set kondisi pada skema yang menjamin sifat tertentu yang berkaitan dengan redundansi dan memperbaiki anomali.

Normal Bentuk Pertama (1NF)

- Jika dan hanya jika semua kolomnya atomik (tidak ada kelompok berulang atau *composit*)
- Setiap *record* harus unik (tidak ada pengulangan)
- Tidak ada atribut yang muncul berulang atau atribut bernilai ganda
- Tiap atribut hanya memiliki satu pengertian.

Normal Bentuk Kedua (2NF)

- Telah memenuhi bentuk normal pertama
- Atribut bukan kunci harus bergantung penuh secara fungsional pada *candidate key* (CK)

Normal Bentuk Ketiga (3NF)

- Telah memenuhi bentuk normal kedua
- Semua atribut bukan himpunan bagian CK tidak mempunyai hubungan yang transitif antar kolom dengan CK.

Contoh :

CK = AB

AB ⊢ C

AB ⊢ D

CD ⊢ A

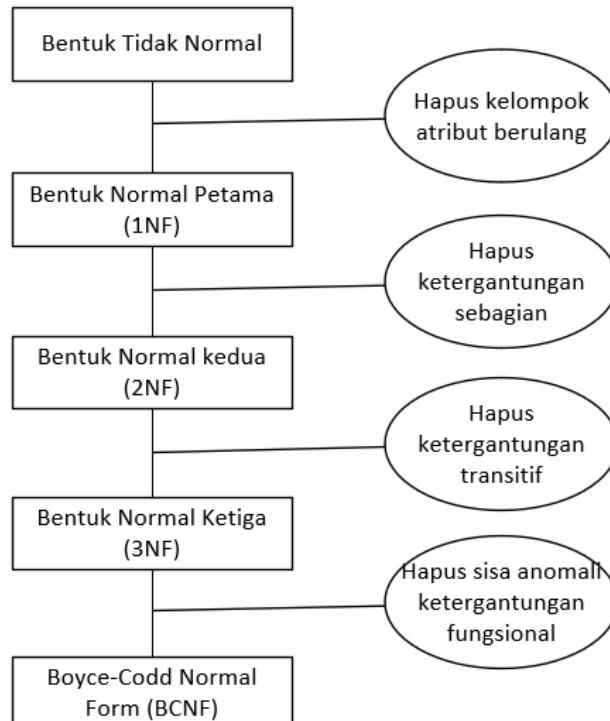
hal ini tidak membuat penggagalan terhadap 3NF, karena A merupakan himp bagian dari CK

CD ⊢ E

hal ini membuat jadi tidak 3NF, karena E bukan merupakan himp bagian dari CK

BCNF

- Telah memenuhi bentuk normal ketiga
- Setiap determinan harus menjadi *candidate key* (CK). Dengan kata lain, setiap ketergantungan fungsional yang terbentuk, ruas kirinya merupakan CK



Gambar 7-1 Tahap normalisasi[8]

Perhatikan Tabel 7-1 di bawah ini:

Tabel 7-1 Mahasiswa

NIM	Nama Mhs	Almt Mhs	JK	kodeMk	namaMK	SKS	Sem	Nilai Mutu	Kode Dosen	Nama Dosen
1301140017	Mark Ruffalo	Jl. Bulan No.19	L	KUG1C3	Dasar Algoritma dan Pemrograman	3	1	B	OKE	Okhe
				CSH2C3	Pemodelan Basisdata	3	4	AB	OKE	Okhe
				SEH2A3	Pemrograman Berorientasi Objek	3	4	E	APA	Aprilia Ananda
1301140137	Chris Evans	Jl. Komet No. 07	L	CSH2C3	Pemodelan Basisdata	3	4	A	OKE	Okhe
				CSH3A4	Jaringan Komputer	4	5	C	HEH	Heru Hasyim
1301140997	Gal Gadot	Jl. Angkasa No. 88	P	KUG1C3	Dasar Algoritma dan Pemrograman	3	1	B	OKE	Okhe
				CSH2C3	Pemodelan Basisdata	3	4	D	OKE	Okhe

Grup Berulang

Tabel di atas belum memenuhi karakteristik relasi dalam konsep Normalisasi karena masih terdapat grup berulang. Sehingga tabel di atas dapat dikatakan belum memenuhi syarat bentuk Normal ke-1. Agar dapat memenuhi bentuk Normal ke-1, susunan tabel harus diubah sebagai berikut:

Tabel 7-2. Tabel Normal 1

NIM	Nama Mhs	Almt Mhs	JK	kodeMk	namaMK	SKS	Sem	Nilai Mutu	Kode Dosen	Nama Dosen
1301140017	Mark Ruffalo	Jl. Bulan No.19	L	KUG1C3	Dasar Algoritma dan Pemrograman	3	1	B	OKE	Okhe
1301140017	Mark Ruffalo	Jl. Bulan No.19	L	CSH2C3	Pemodelan Basisdata	3	4	AB	OKE	Okhe
1301140017	Mark Ruffalo	Jl. Bulan No.19	L	SEH2A3	Pemrograman Berorientasi Objek	3	4	E	APA	Aprilia Ananda
30112777	Chris Evans	Jl. Komet No. 07	L	CSH2C3	Pemodelan Basisdata	3	4	A	OKE	Okhe
30112777	Chris Evans	Jl. Komet No. 07	L	CSH3A4	Jaringan Komputer	4	5	C	HEH	Heru Hasyim
30112999	Gal Gadot	Jl. Angkasa No. 88	P	KUG1C3	Dasar Algoritma dan Pemrograman	3	1	B	OKE	Okhe
30112999	Gal Gadot	Jl. Angkasa No. 88	P	CSH2C3	Pemodelan Basisdata	3	4	D	OKE	Okhe

Tahap selanjutnya menetapkan ketergantungan antar atribut (*functional dependency*).

1. Tentukan mana *Candidate Key* dari tabel tersebut.

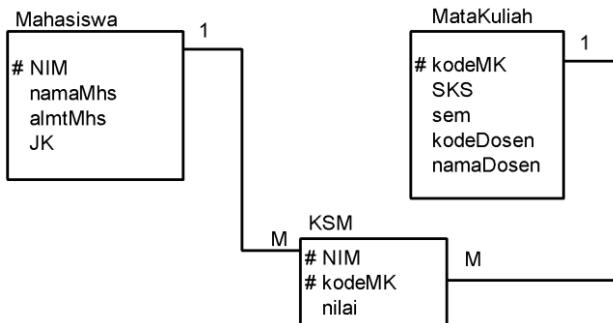
Dari tabel di atas, tidak ada atribut tunggal yang dapat dijadikan sebagai *Candidate Key*. Karena itu pastilah *Candidate Key* merupakan *composite key*.

Candidate Key adalah: **NIM, kodeMK**

INGAT! *Candidate key* pastilah merupakan determinan. Dia dapat menentukan nilai setiap atribut pada tabel.

- Nim, kodeMK → namaMhs
- Nim, kodeMK → almtMhs

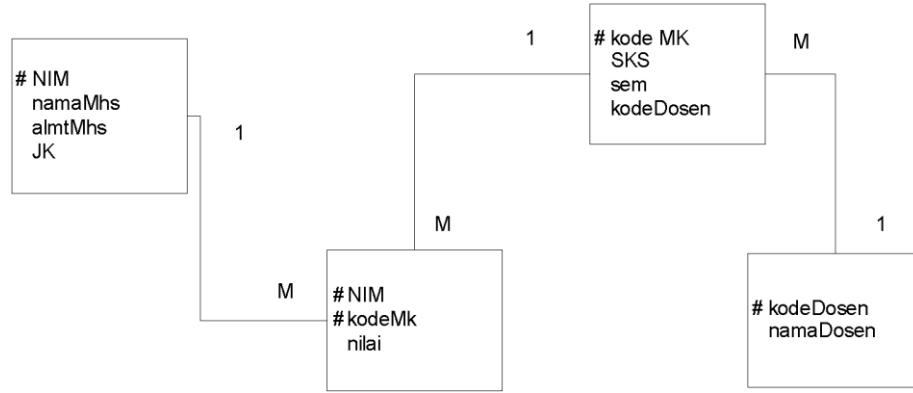
- Nim, kodeMK → JK
 - Nim, kodeMK → namaMK
 - Nim, kodeMK → SKS
 - Nim, kodeMK → sem
 - Nim, kodeMK → nilaiMutu
 - Nim, kodeMK → kodeDosen
 - Nim, kodeMK → namaDosen
2. Tentukan atribut-atribut yang memiliki ketergantungan sebagian terhadap *candidate key*.
- Tabel di atas memiliki *Candidate Key* yang merupakan *composite key*. Cermati untuk tabel-tabel yang *Candidate Key* nya merupakan *composite key* BIASANYA memiliki ketergantungan sebagian (walaupun tidak selalu).
 - Tabel 2 di atas sudah memenuhi bentuk Normal ke-1, tetapi masih belum memenuhi syarat bentuk Normal ke-2 karena masih terdapat ketergantungan sebagian terhadap *candidate key*-nya sebagai berikut:
 - Nim → namaMhs
 - Nim → almtMhs
 - Nim → JK
 - kodeMK → namaMK
 - kodeMK → SKS
 - kodeMK → sem
 - kodeMK → kodeDosen
 - kodeMK → namaDosen
3. Pisahkan atribut-atribut yang memiliki ketergantungan sebagian menjadi tabel-tabel yang berdiri sendiri. Sehingga tabel yang terbentuk sebagai berikut



Gambar 7-2 Relasi Antar Tabel

4. Periksa kembali setiap tabel yang terbentuk apakah sudah memenuhi syarat *Fully Functional Dependency* (atribut bukan *Candidate Key* sudah tergantung sepenuhnya pada *candidate Key*)
- Dari gambar 1 di atas, Tabel Mahasiswa & Tabel KSM sudah memenuhi syarat *fully functional dependency*, tetapi tidak untuk Tabel Matakuliah. (Tabel Mahasiswa & Tabel KSM sampai tahap 3NF sudah NORMAL).
 - Tabel Matakuliah masih belum memenuhi syarat *fully functional dependency* karena masih terdapat ketergantungan transitif (belum memenuhi bentuk Normal ke-3) sebagai berikut:
 $\text{kodeMK} \rightarrow \text{kodeDosen}$
 $\text{kodeDosen} \rightarrow \text{namaDosen}$

- INGAT! Ketergantungan transitif adalah dimana terdapat atribut yang bukan *candidate key* tergantung pada atribut yang bukan *candidate key* lainnya dalam satu tabel.
- Asumsi: kodeMK menentukan kodeDosen karena 1 matakuliah hanya diajarkan oleh 1 dosen.
- Solusi untuk masalah ini adalah DEKOMPOSISI TABEL. Pisahkan atribut-atribut yang memiliki ketergantungan transitif menjadi tabel yang berdiri sendiri. Sehingga tabel yang terbentuk adalah sebagai berikut



Gambar 7-3 Relasi Antar Tabel Final

Relasi antar tabel di atas (gambar 2) sudah Normal. Keempat tabel tersebut juga telah memenuhi BCNF, karena KF dari setiap tabel, ruas kirinya merupakan CK.

7.2. Studi Kasus

Berdasarkan studi kasus yang terdapat pada modul 1 dan 2, Anda diminta untuk memperbaiki struktur tabel berdasarkan kebutuhan bisnis terbaru dari bioskop. Adapun desain struktur *database* terbaru dibuat berdasarkan tabel-tabel universal yang diberikan di bawah ini:

Tabel 7-3 Tabel Identitas Film

	<h2>Spider-Man: Far From Home(2019)</h2> <p>Sinopsis:</p> <p>Spider-Man Far From Home menjadi film marvel pertama setelah Avengers: Endgame. Film ini menjadi sekuel lanjutan dari fil Spider Man Homecoming yang dirilis 5 Juli 2017 silam. Jon Watts didapuk menjadi sutradara dalam film ini, sedangkan skenarionya ditulis oleh Chris McKenna dan Erik Sommers. Aktor Tom Holland masih didapuk sebagai pemeran utama dalam film ini.</p> <p>Dalam Spider-Man: Far From Home sebagian akan berurusan dengan bagaimana Peter menangani Decimation dan fakta bahwa Tony Stark sekarang sudah mati. Seperti yang ditunjukkan trailer terbaru, Peter akan dihadapkan dengan kematian Tony di setiap kesempatan. Dia tidak akan punya banyak waktu untuk bersedih, karena Spider-Man akan direkrut oleh Nick Fury untuk menangani beberapa urusan yang mendesak. Ternyata, Spider-Man direkrut untuk bertarung melawan makhluk kuno yang dikenal sebagai Elementals, yang meliputi Hydron, Hellfire, Magnum, dan Zephyr. Saat bertarung dengan Elementals, Spider-Man akan dibantu oleh Mysterio, karakter pemegang sihir yang dimainkan oleh Jake Gyllenhaal. Tetapi karakter baru Gyllenhaal yang misterius mungkin memiliki motif tersembunyi.</p> <p>Siapakah Mysterio? Dia adalah karakter yang memiliki pengetahuan dan keterampilan yang hebat untuk menciptakan ilusi yang meyakinkan dan efek khusus. Beberapa orang berspekulasi bahwa Mysterio adalah seorang mistikus seperti Doctor Strange, meskipun dia tidak setinggi itu. Dalam kasus ini Mysterio gagal sebagai seorang mistikus tetapi masih berpegang teguh pada harapan bahwa dia bisa menjadi pahlawan. Jadi dia menggunakan ilusi, yang merupakan keahlian terbaiknya. Tentu saja, Mysterio bisa saja menggunakan gadget yang menunjang penampilannya. Intinya adalah, dia mungkin bertindak seperti pahlawan, tetapi dia akan terungkap sebagai penjahat.</p> <p>Genre : Action, Adventure, Sci-Fi</p> <p>Sutradara : Jon Watts</p> <p>Penulis : Chris McKenna, Stan Lee</p> <p>Pemain Film :</p> <ul style="list-style-type: none">● Tom Holland sebagai (Peter Parker/Spider-Man)● Samuel L. Jackson sebagai (Nick Fury)● Jake Gyllenhaal sebagai (Quentin Beck / Mysterio)● Marisa Tomei sebagai (May Parker)● Jon Favreau sebagai (Happy Hogan)● Zendaya. sebagai (MJ)● Jacob Batalon sebagai (Ned Leeds)● Tony Levolori sebagai (Flash Thompson)● Angourie Rice sebagai (Betty Brant)● Remy Hii sebagai (Brad Davis)● Martin Starr sebagai (Mr. Harrington)● J.B. Smoove sebagai (Mr. Dell) <p>Rumah Produksi : Marvel Entertainment, Marvel Studios</p> <p>MPAA : PG-13</p> <p>Durasi : 2h 9min</p>
---	--

Tabel 7-4 Tabel Rekap Transaksi

Kode transaksi	Kode Petugas	ID Member	Judul film	Tanggal Tayang	Jam Tayang	Tanggal Pembelian	Jam Pembelian	Teater	Nomor Kursi		Harga	Pembayaran
									Baris	Kolom		
63653041600413	FAD	26240	Captain America: Civil War	Selasa, 26 April 2016	19.00	26-Apr-16	01.22.55	3	D	4	60.000	VC
89393041600116	YUN	34951	Batman Vs Superman: Dawn of Justice	Rabu, 6 April 2016	21.15	Rabu, 6 April 2016	19.32.47	5	A	12	35.000	DC
89393041600216			5					A	13	DC		
89393041600316			5					A	14	DC		
15295061600111	YUN	150583	Now You See Me 2	Kamis, 30 Juni 2016	15.50	Kamis, 30 Juni 2016	14.30.16	2	B	7	35.000	Cash
15295061600211			2					B	8	Cash		

Keterangan kode pembayaran:

VC : Voucher cash. Pembayaran dengan menggunakan Voucher dari merchant lain, seperti T-Cash, voucher Ponta, voucher Sepulsa, dll.

DC : Deposit cash. Pembayaran dengan menggunakan Deposit uang yang dimiliki oleh member.

Cash : Pembayaran secara tunai melalui transfer ATM atau *internet banking*.

Anda diminta untuk melakukan normalisasi data!

7.3. Tutorial

1. Baca kembali Tabel 3 dan Tabel 4.
2. Identifikasi fakta-fakta apa saja yang harus disimpan di dalam *database* dan yang saat ini belum tersimpan dalam *database*.
3. Asumsikan semua fakta tersebut menjadi atribut-atribut yang akan disimpan dalam *database*. Buat listnya.
4. Jika terdapat grup berulang, maka jadikan sebagai data yang atomik.
5. Identifikasi ketergantungan fungsional (KF) antar atribut.
6. Tentukan *candidate key* (CK) dari tabel.
7. Tentukan atribut yang tidak memiliki ketergantungan penuh (ketergantungan sebagian) terhadap CK.
8. Pisahkan atribut-atribut yang memiliki ketergantungan sebagian menjadi tabel-tabel yang berdiri sendiri.
9. Tentukan atribut yang memiliki ketergantungan transitif.
10. Pisahkan atribut-atribut yang memiliki ketergantungan transitif menjadi tabel-tabel yang berdiri sendiri.
11. Periksa kembali setiap tabel yang terbentuk apakah sudah memenuhi syarat *Fully Functional Dependency* (atribut bukan CK sudah tergantung sepenuhnya pada CK).
12. Periksa setiap KF yang ada saat ini. Untuk membuat tabel menjadi BCNF, maka setiap ruas kiri dari KF harus merupakan CK.
13. Identifikasi hubungan antara entitas baru yang terbentuk dengan entitas yang sudah ada sebelumnya.
14. Periksa kembali, apakah perlu ada perubahan atribut pada entitas atau relasi lama. Buat kembali struktur *database* sesuai dengan desain *logical* yang telah terbentuk.

Modul 8. Overview NoSQL

Tujuan Praktikum

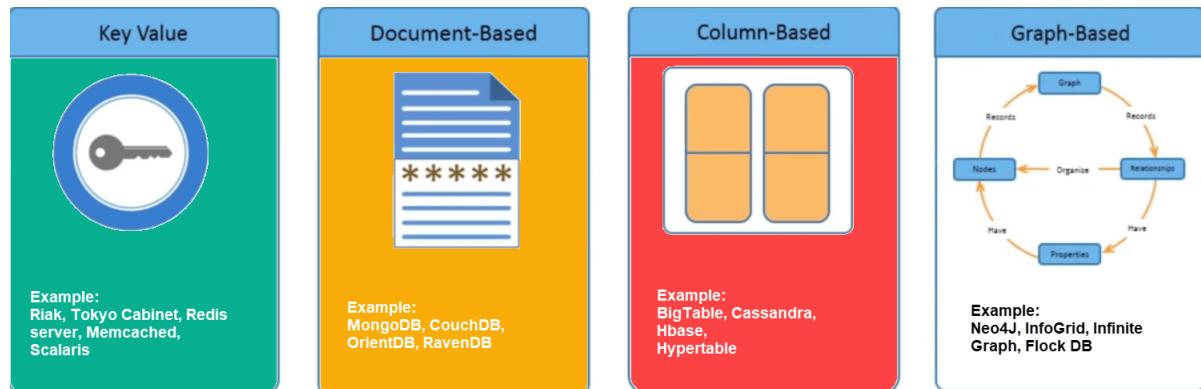
- Praktikan mampu memahami tools database non-relasional

8.1. Ringkasan Materi

Basis Data NoSQL adalah Sistem Manajemen Basis Data non-relasional, yang tidak memerlukan skema tetap. Basis data jenis ini tidak memerlukan join, dan mudah untuk diskalakan. Tujuan utama penggunaan database NoSQL adalah untuk penyimpanan data terdistribusi dengan kebutuhan penyimpanan data yang sangat besar. NoSQL digunakan untuk Big data dan aplikasi web real-time. Misalnya, perusahaan seperti Twitter, Facebook, dan Google mengumpulkan terabyte data pengguna setiap hari.

Basis data NoSQL adalah singkatan dari "Not Only SQL" atau "Not SQL." Carl Strozz memperkenalkan konsep NoSQL pada tahun 1998. RDBMS tradisional menggunakan sintaks SQL untuk menyimpan dan mengambil data untuk penyediaan informasi. Sebaliknya, sistem basis data NoSQL mencakup berbagai teknologi basis data yang dapat menyimpan data terstruktur, semi-terstruktur, tidak terstruktur, dan polimorfik.

8.1.1. Jenis Basis Data NoSQL



1. Key-Value

Data disimpan dalam pasangan key/value, dan disimpan sebagai array byte. Key-Value Database memiliki performa tinggi, sangat skalabel, sangat fleksibel, dan kompleksitas rendah. Database penyimpanan pasangan nilai kunci menyimpan data sebagai tabel hash di mana setiap kunci unik, dan nilainya bisa berupa JSON, BLOB (Binary Large Objects), string, dll.

Database NoSQL semacam ini digunakan sebagai koleksi, kamus, array asosiatif, dll. Penyimpanan nilai kunci membantu pengembang untuk menyimpan data tanpa skema. Basis data jenis ini paling baik untuk konten keranjang belanja (shopping cart).

Misalnya: Amazon DynamoDB, Cassandra, Voldemort, RAMCloud, dan Flare.

Contoh:

Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

2. Column-based

Database berorientasi kolom (Column-oriented) bekerja pada kolom dan didasarkan pada BigTable oleh Google. Setiap kolom diperlakukan secara terpisah. Nilai database kolom tunggal disimpan secara berurutan.

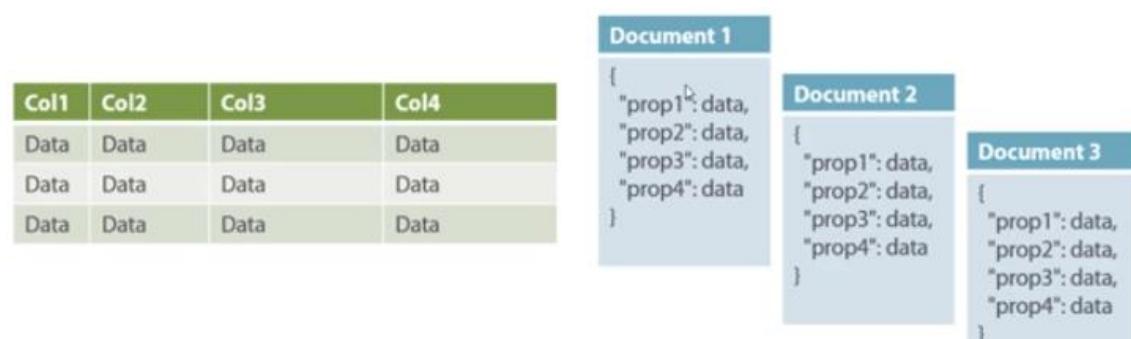
Basis data jenis ini memberikan kinerja tinggi pada kueri agregasi seperti SUM, COUNT, AVG, MIN, dll. Hal ini dikarenakan data sudah tersedia dalam kolom. Basis data NoSQL berbasis kolom banyak digunakan untuk mengelola gudang data, intelijen bisnis, CRM, katalog kartu perpustakaan.

Contoh:

ColumnFamily			
Row Key	Column Name		
	Key	Key	Key
Value			
	Value	Value	Value
Column Name			
	Key	Key	Key
	Value	Value	Value

3. Document-oriented

Database NoSQL Berorientasi Dokumen menyimpan dan mengambil data sebagai pasangan nilai kunci, tetapi bagian nilai disimpan sebagai dokumen. Dokumen disimpan dalam format JSON atau XML. Nilai dapat dibuat query.



DB Relasional vs DB Document-oriented

Pada database relasional terdapat baris dan kolom, dan pada database dokumen terdapat struktur yang mirip dengan JSON. Untuk database relasional, kita perlu mengetahui kolom apa

yang terdapat pada table dan seterusnya. Namun, untuk database dokumen terdapat penyimpanan data seperti objek JSON.

Jenis DB berorientasi dokumen sebagian besar digunakan untuk sistem CMS, platform blogging, analitik real-time & aplikasi e-commerce. DB jenis ini sebaiknya tidak digunakan untuk transaksi kompleks yang membutuhkan banyak operasi atau kueri terhadap berbagai struktur agregat.

4. Graph-based

Database tipe grafik menyimpan entitas serta relasi di antara entitas tersebut. Entitas disimpan sebagai node dengan relasi sebagai edge. Edge memberikan hubungan antar node. Setiap node dan edge memiliki pengidentifikasi unik. Database Grafik bersifat multi-relasional, sebagian besar digunakan untuk jejaring sosial, logistik, data spasial.

Modul 9. NoSQL – Key Value Database

Tujuan Praktikum

- Praktikan mampu membuat database non relasional dengan jenis Key-Value Database

9.1. Ringkasan Materi

9.1.1. Apache Cassandra

Apache Cassandra (<http://cassandra.apache.org/>) adalah sistem manajemen database NoSQL terdistribusi free dan open-source yang dirancang untuk menangani data dalam jumlah besar di banyak layanan komoditas, bersifat high availability.

Cassandra menawarkan dukungan kuat untuk klaster yang menjangkau beberapa pusat data, dengan mereplikasi tanpa master asinkron yang memungkinkan operasi latensi rendah untuk semua klien. Cassandra memiliki struktur yang bersifat highly scalable, possibly consistent, dan terdistribusi.

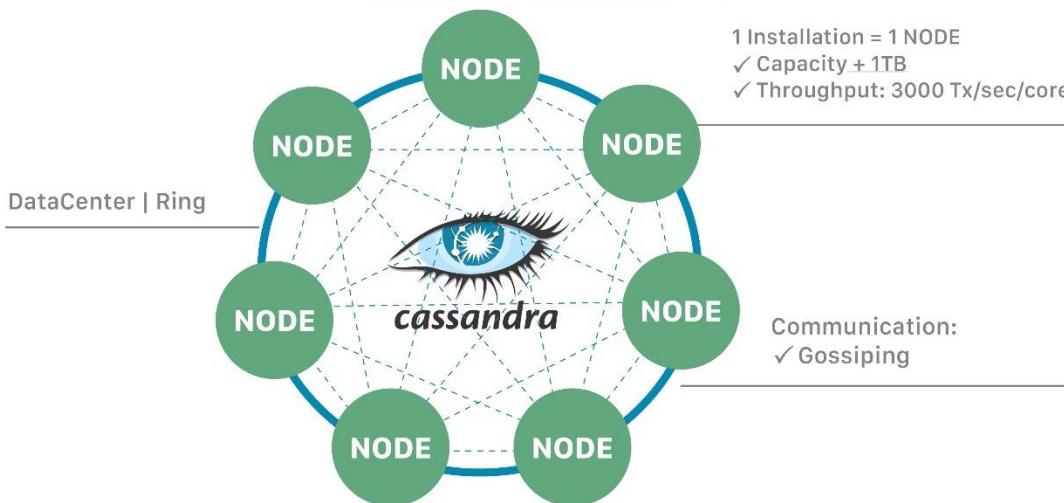
Keuntungan Cassandra untuk pengembangan web

- Cassandra dikembangkan menjadi server terdistribusi, tetapi juga dapat dijalankan sebagai node sederhana
- Skalabilitas horizontal (dapat menambahkan perangkat keras baru bila perlu)
- Melayani penyediaan data cepat bahkan jika permintaan meningkat.
- Kecepatan tulis tinggi untuk mengelola volume data tambahan.
- Penyimpanan terdistribusi.
- Kemampuan untuk mengubah struktur data saat pengguna menuntut lebih banyak fungsi.
- API sederhana dan bersih untuk bahasa pemrograman favorit Anda.
- Deteksi kesalahan otomatis (toleransi kesalahan).
- Terdesentralisasi.
- Mengizinkan penggunaan Hadoop untuk menggunakan Pengurangan Peta.

Kekurangan:

- Tanpa Kueri Ad-Hoc: Perlu "memodelkan" data pada kueri yang ingin di tampilkan, bukan pada struktur data itu sendiri.
- Tanpa Agregasi: Karena Cassandra adalah penyimpanan key-value, operasi seperti SUM, MIN, MAX, AVG, dan agregasi lainnya membutuhkan sumber daya intensif untuk diselesaikan.
- Performa yang Tidak Dapat Diprediksi: Karena Cassandra memiliki banyak job asinkron dan task berbeda yang tidak di schedule oleh pengguna, performanya tidak dapat diprediksi.

ApacheCassandra™= NoSQL Distributed Database



Salah satu atribut penting Cassandra adalah basis datanya terdistribusi. Basis data Cassandra dengan mudah diskalakan saat aplikasi berada di bawah tekanan tinggi, dan distribusinya juga mencegah kehilangan data dari kegagalan perangkat keras pusat data mana pun. Arsitektur terdistribusi juga menghadirkan kekuatan teknis; misalnya, pengembang dapat men-tweak throughput kueri baca atau menulis kueri secara terpisah.

"Terdistribusi" berarti bahwa Cassandra dapat berjalan di beberapa mesin sambil tampil kepada pengguna sebagai satu kesatuan. Karena ini adalah database terdistribusi, Cassandra dapat (dan biasanya) memiliki banyak node. Sebuah node mewakili satu instance dari Cassandra. Node-node ini berkomunikasi satu sama lain melalui protokol yang disebut gosip, yang merupakan proses komunikasi peer-to-peer komputer. Cassandra juga memiliki arsitektur tanpa master – node mana pun dalam database dapat memberikan fungsionalitas yang sama persis dengan node lainnya – berkontribusi pada ketahanan dan ketahanan Cassandra. Beberapa node dapat diatur secara logis ke dalam sebuah cluster, atau "cincin". Sehingga memungkinkan dapat memiliki beberapa pusat data.

9.1.2. Data Model

1. Column

Pada Cassandra, unit penyimpanan dasar adalah column. Column adalah tupel yang berisi nama kolom, nilai, dan timestamp. Timestamp menyimpan waktu pembaruan kolom dan digunakan untuk resolusi konflik. Nama kolom analog dengan nama atribut dalam tabel dalam database relasional.

2. Keyspace

Keyspace adalah wadah yang menyimpan data yang digunakan aplikasi. Keyspace dapat berasosiasi dengan satu atau lebih kelompok kolom. Keyspace mengharuskan beberapa atribut didefinisikan, seperti nama yang ditentukan pengguna, strategi replikasi, dan lainnya.

Keyspace analog dengan database dalam RDBMS tetapi tanpa interrelations.

Atribut dasar yang dapat dikaitkan dengan keyspace adalah:

- Replication factor (Faktor replikasi): berapa banyak yang ingin di bayarkan dalam kinerja demi konsistensi.

- Strategi penempatan replika: menunjukkan bagaimana replika ditempatkan pada ring (SimpleStrategy, OldNetworkTopologyStrategy, dan NetworkTopologyStrategy)
- Column Families: setidaknya satu per Keyspace adalah wadah baris, berisi kolom.

3. Cassandra Query Language (CQL)

CQL menawarkan model yang mirip dengan SQL dalam arti bahwa data disimpan dalam tabel yang berisi baris dan kolom. Oleh karena itu, pada CQL, istilah-istilah seperti tabel, baris, dan kolom, memiliki definisi yang sama dengan yang ada di SQL.

Beberapa fitur yang dimiliki adalah:

Data Types, Data Definition, Data Manipulation, Secondary Indexes, Materialized Views, Security, Functions, Arithmetic Operators, JSON Support, Triggers.

Column: terdiri dari name, value, dan timestamp.

Cluster: mesin yang membentuk instance (turunan) dari Cassandra. Dapat berisi beberapa Keyspace.

Keyspace: namespace untuk sekumpulan ColumnFamily yang terkait dengan aplikasi.

ColumnFamily: berisi banyak columns. Pada model relasional diasosiasikan dengan table.

SuperColumn: columns yang memiliki sub-columns.

Contoh aplikasi: Twissandra-j

Aplikasi Java yang menggunakan Cassandra dan CQL 3.0. Ini menyajikan web yang mirip dengan twitter, didasarkan pada contoh terkenal Twissandra di Python dan Django di Cassandra:

<https://github.com/twissandra/twissandra>

9.2. Tutorial

1. Download Apache Cassandra dari <http://cassandra.apache.org/download/>
2. Install Apache Cassandra (cek video instalasi
<https://www.youtube.com/watch?v=NF8OEVgV3bk>)
3. Lakukan koneksi Cassandra menggunakan CQL

```
$ cqlsh -u <username> -p <password> localhost
```
4. Buat Keyspace bernama **test** dengan faktor replikasi 1 dan simple strategy.

```
CREATE KEYSPACE test WITH
    REPLICATION = {'class' : 'SimpleStrategy', 'replication_factor': 1};
```
5. Gunakan keyspace test

```
Use test;
```
6. Buat column family (table) bernama **person** dengan atribut: id *text*, email *text*, name *text*, surname *text* dan primary key-nya id.

```
CREATE TABLE person (
    id text,
    email text,
    name text,
    surname text,
    PRIMARY KEY (id));
```
7. Periksa skema dari column family **person**.

```
DESCRIBE person;
```

8. Tambahkan data person pada column family.
 - `INSERT INTO person (id, name, surname, email) VALUES ('001', 'Shalabh', 'Aggarwal', 'contact@shalabhaggarwal.com');`
 - `INSERT INTO person (id, name, surname, email) VALUES ('002', 'John', 'Doe', 'john@example.com');`
 - `INSERT INTO person (id, name, surname, email) VALUES ('003', 'Harry', 'Potter', 'harry@example.com');`
9. Tampilkan semua data dari person
`SELECT * FROM person;`
10. Tampilkan data person dengan id = 001
`SELECT name FROM person WHERE id='001';`
11. Menggunakan set. Set merupakan kumpulan value. Value disimpan sebagai tidak terurut, tetapi CQLSH akan mengembalikan value dengan cara yang disortir. Misalnya, string akan diurutkan menurut abjad.
 Buat column family **users** dengan perintah columnfamily yang memiliki columns key varchar PRIMARY KEY, full_name varchar, birth_date int, state varchar, dan set text bernama emails **emails set<text>**
`CREATE COLUMNFAMILY users (
 key varchar PRIMARY KEY,
 full_name varchar,
 birth_date int,
 state varchar,
 emails set<text>);`
12. Buatlah index pada users pada kolom birth_date dan state.
`CREATE INDEX ON users (birth_date);
CREATE INDEX ON users (state);`
13. Isi columnfamily users dengan data sebagai berikut:
 'pangeles', 'Pilar Angeles', 1975, 'UT', 'plang@gmail.com', mpa@hotmail.com'
 'asmith', 'Alice Smith', 1973, 'WI', asmith@Gmail.com'
 'htayler', 'Howard Tayler', 1968,
 'UT', {'hty@Hotmail.com', 'otheremail@Gmail.com'}

 - `INSERT INTO users (key, full_name, birth_date, state, emails) VALUES ('pangeles', 'Pilar Angeles', 1975, 'UT', {'plang@gmail.com', mpa@hotmail.com});`
 - `INSERT INTO users (key, full_name, birth_date, state) VALUES ('asmith', 'Alice Smith', 1973, 'WI', {asmith@Gmail.com});`
 - `INSERT INTO users (key, full_name, birth_date, state) VALUES ('htayler', 'Howard Tayler', 1968, 'UT', {'hty@Hotmail.com', 'otheremail@Gmail.com'});`

14. Tampilkan full name dan emails dari Pilar Angeles
`Select full_name, emails from users where key= 'pangeles'`
15. Tampilkan key dan state dari users
`SELECT key, state FROM users;`
16. Tampilkan semua data users yang tinggal di UT dan lahir setelah tahun 1970
`SELECT * FROM users WHERE state='UT' AND birth_date > 1970 ALLOW FILTERING;`

Modul 10. NoSQL – Columnar Database

Tujuan Praktikum

- Praktikan mampu membuat database non relasional dengan jenis Columnar Database

10.1. Ringkasan Materi

Columnar database disebut juga database biner-relasional (binary-relational database). Model Biner-Relasional berawal pada tahun 1970-an. Chen adalah orang pertama yang menyebutkannya dalam peran transendentalnya " Entity Relational Modeling - towards a unified view of data". Model relasional didasarkan pada konsep matematika dari relationship.

Model Relasional-biner adalah model khusus dari model relasional, dengan batasan bahwa semua relasi memiliki derajat 2. Contoh columnar database: Sybase IQ, ParAccel, Vertica, MonetDB, SAND Technologies.

Columnar database digunakan dalam sistem manajemen basis data (DBMS) yang membantu menyimpan data dalam kolom daripada baris. Hal ini dapat mempercepat waktu yang diperlukan untuk mengembalikan kueri tertentu. Proses penyimpanan secara kolom juga meningkatkan kinerja I/O disk, sangat membantu dalam analitik data dan data warehouse. Tujuan utama dari Columnar Database adalah untuk membaca dan menulis data secara efektif.

10.1.1. Columnar Database vs Rows Database

Basis data Kolom dan Baris adalah beberapa metode yang digunakan untuk memproses big data analitik dan data warehouse. Tetapi pendekatan mereka berbeda satu sama lain.

Berikut adalah contoh tabel database sederhana dengan empat kolom dan tiga baris.

ID Number	Last Name	First Name	Bonus
534782	Miller	Ginny	6000
585523	Parker	Peter	8000
479148	Stacy	Gwen	2000

Dalam Columnar database, data yang disimpan menggunakan format sebagai berikut:

534782, 585523, 479148; Miller, Parker, Stacy; Ginny, Peter, Gwen; 6000, 8000, 2000

Dalam DBMS berorientasi Baris, data disimpan dalam format berikut:

534782, Miller, Ginny, 6000; 585523, Parker, Peter, 8000; 479148, Stacy, Gwen, 2000

Columnar database sangat baik digunakan ketika:

- Kueri yang hanya melibatkan beberapa kolom.
- Kompresi tetapi hanya berdasarkan kolom.
- Mengelompokkan kueri terhadap sejumlah besar data.

Keuntungan dari Columnar Database:

1. Columnar database dapat digunakan untuk tugas yang berbeda seperti pada aplikasi yang berkaitan dengan big data.
2. Data dalam columnar database memiliki sifat yang sangat dapat dimampatkan dan memiliki operasi yang berbeda seperti (AVG), (MIN, MAX), yang diizinkan oleh kompresi.
3. Efisiensi dan Kecepatan: Kecepatan kueri Analitik yang dilakukan lebih cepat.
4. Pengindeksan sendiri: Manfaat lain dari DBMS berbasis kolom adalah pengindeksan sendiri, yang menggunakan lebih sedikit ruang disk daripada sistem manajemen basis data relasional yang berisi data yang sama.

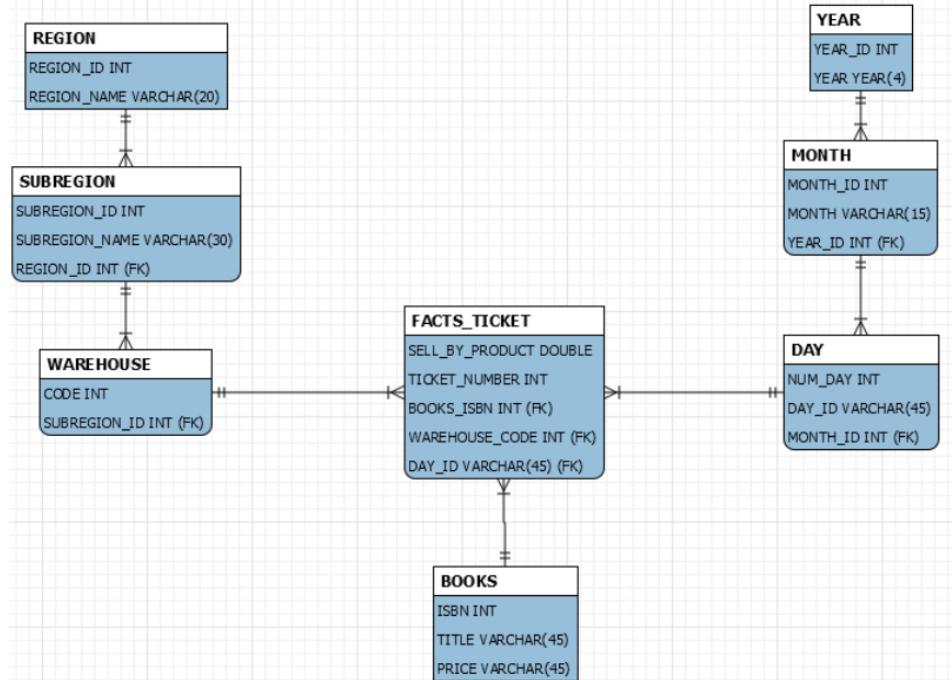
Batasan Columnar database:

1. Untuk memuat data inkremental, basis data tradisional lebih relevan dibandingkan dengan basis data berorientasi kolom.
2. Untuk aplikasi pemrosesan transaksi online (OLTP), basis data berorientasi baris lebih tepat daripada basis data kolumnar.

10.2. Tutorial

Menerapkan sistem OLAP ke dalam database berbentuk kolom (columnar database) menggunakan SQL.

1. Unduh SAP IQ (free trial). (<https://www.sap.com/products/technology-platform/sybase-iq-big-data-management/trial.html>)
2. Gunakan model relasional OLAP berikut:



3. Instal SAP IQ
4. Buat tabel seperti yang diperlihatkan dalam model relasional untuk OLAP pada nomor 2.
Note: SAP IQ menggunakan SQL, jadi Anda dapat menggunakan Kode SQL yang sama untuk mengimplementasikan dan membuat query database Anda
5. Isilah database, agar mampu mengeksekusi query no 6.
6. Tampilkan informasi total penjualan harian berdasarkan produk dan subregion.

Modul 11. NoSQL – Document Database

Tujuan Praktikum

- Praktikan mampu membuat database non relasional dengan jenis Document Database

11.1. Ringkasan Materi

Document database (juga dikenal sebagai database berorientasi dokumen atau penyimpanan dokumen) adalah database yang menyimpan informasi dalam dokumen. Document database adalah jenis database nonrelasional yang dirancang untuk menyimpan dan meminta data sebagai dokumen mirip JSON.

Document database memudahkan pengembang untuk menyimpan dan membuat kueri data dalam database dengan menggunakan format model dokumen yang sama yang mereka gunakan dalam kode aplikasi mereka. Sifat dokumen dan basis data dokumen yang fleksibel, semiterstruktur, dan hierarkis memungkinkannya berkembang sesuai kebutuhan aplikasi.

Model dokumen bekerja dengan baik dengan kasus penggunaan seperti katalog, profil pengguna, dan sistem manajemen konten di mana setiap dokumen unik dan berkembang dari waktu ke waktu. Document database memungkinkan pengindeksan yang fleksibel, kueri ad hoc yang kuat, dan analitik atas kumpulan dokumen.

11.1.1 MongoDB

MongoDB merupakan cross-platform, database berorientasi dokumen yang memberikan kinerja tinggi, ketersediaan tinggi, dan skalabilitas mudah. MongoDB bekerja pada konsep collection dan dokumen.

Beberapa istilah penting:

- Database.** Database adalah wadah fisik untuk collection. Setiap database mendapatkan kumpulan file sendiri di sistem file. Satu server MongoDB biasanya memiliki banyak basis data.
- Collection.** Collection adalah sekelompok dokumen MongoDB. Collection setara dengan tabel pada RDBMS. Collection berada dalam satu database dan tidak menerapkan skema. Dokumen dalam collection dapat memiliki field yang berbeda. Biasanya, semua dokumen dalam collection memiliki tujuan yang serupa atau terkait.
- Document.** Dokumen adalah kumpulan pasangan key-value. Dokumen memiliki skema dinamis. Skema dinamis berarti bahwa dokumen dalam collection yang sama tidak perlu memiliki rangkaian bidang atau struktur yang sama, dan bidang umum dalam dokumen kumpulan dapat menyimpan jenis data yang berbeda.

Tabel berikut menunjukkan hubungan terminologi RDBMS dengan MongoDB:

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
Column	Field
Table Join	Embedded Documents

Primary Key	Primary Key (Default key <code>_id</code> provided by mongodb itself)
-------------	---

11.1.2 MongoDB Advantages

Setiap basis data relasional memiliki desain skema tipikal yang menunjukkan jumlah tabel dan hubungan antara tabel-tabel ini. Namun pada MongoDB, tidak ada konsep relasi.

Keuntungan MongoDB dibandingkan RDBMS:

1. Tanpa skema: MongoDB adalah basis data dokumen di mana satu collection menyimpan dokumen yang berbeda. Jumlah bidang, konten, dan ukuran dokumen dapat berbeda dari satu dokumen ke dokumen lainnya.
2. Struktur objek tunggal jelas.
3. Tidak ada Join kompleks.
4. Deep query-ability. MongoDB mendukung kueri dinamis pada dokumen menggunakan bahasa kueri berbasis dokumen yang hampir sekuat SQL.
5. Tuning.
6. Kemudahan scale-out: MongoDB mudah untuk di skalakan.
7. Konversi/pemetaan objek aplikasi ke objek database tidak diperlukan.
8. Menggunakan memori internal untuk menyimpan perangkat kerja (windowed), memungkinkan akses data yang lebih cepat.

11.2 Tutorial 1

1. Unduh MongoDB (www.mongodb.com)
2. Install mongodb (Screenshot setiap langkahnya)
 - a. Jalankan instalasi sebagai administrator.
Default path instalasi adalah: C:\ Program Files \ MongoDB \ Server \ 3.4
Binary yang diinstall adalah:
 - Server mongod.exe
 - Router mongos.exe Client mongo.exe
 - MonitoringTools mongostat.exe, mongotop.exe
 - ImportExportTools mongodump.exe, mongorestore.exe, mongoexport.exe, mongoimport.exe MiscellaneousTools bsondump.exe, mongofiles.exe, mongooplog.exe, mongoperf.exe
 - b. Buat direktori: **mkdir** pada: C:\ Program Files \ MongoDB \ data
 - c. Buat subdirektori pada data sebagai db untuk menyimpan objek database. (secara default tersimpan pada root c :; ubah path tersebut dengan parameter **-dbpath**)
 - d. Buat subdirectory: **mkdir** pada C:\ Program Files \ MongoDB \ data \ log, yang akan digunakan sebagai blog
3. Start MongoDB
 - a. Temukan corresponding binary di C:\ Program Files \ MongoDB \ Server \ 3.4 \ bin
 - b. Disarankan membuat shortcut untuk mongod.exe
 - c. Run server binary sebagai administrator:
 "C:\ Program Files \ MongoDB \ Server \ 3.4 \ bin \ mongod.exe" --dbpath "C:\ Program Files \ MongoDB \ data \ db"

```

C:\Program Files\MongoDB\Server\3.4\bin>mongod.exe --dbpath "c:\Program Files\MongoDB\data\db"
2017-07-06T11:15:29.124-0500 I CONTROL [initandlisten] MongoDB starting : pid=8708 port=27017 dbpath=c:\Program Files\MongoDB\data\db 64-bit host=DESKTOP-VJMKRDU
2017-07-06T11:15:29.125-0500 I CONTROL [initandlisten] targetMinOS: Windows Vista/Windows Server 2008
2017-07-06T11:15:29.127-0500 I CONTROL [initandlisten] db version v3.4.6
2017-07-06T11:15:29.128-0500 I CONTROL [initandlisten] git version: c55eb86ef46ee7aede3b1e2a5d184a7df4fbfb5b5
2017-07-06T11:15:29.129-0500 I CONTROL [initandlisten] allocator: tcmalloc
2017-07-06T11:15:29.130-0500 I CONTROL [initandlisten] modules: none
2017-07-06T11:15:29.130-0500 I CONTROL [initandlisten] build environment:
2017-07-06T11:15:29.130-0500 I CONTROL [initandlisten]   distarch: x86_64
2017-07-06T11:15:29.131-0500 I CONTROL [initandlisten]   target_arch: x86_64
2017-07-06T11:15:29.131-0500 I CONTROL [initandlisten] options: { storage: { dbPath: "c:\Program Files\MongoDB\data\db" } }
2017-07-06T11:15:29.134-0500 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=3420M,session_max=2000,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=10000),checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0),
2017-07-06T11:15:29.261-0500 I CONTROL [initandlisten]
2017-07-06T11:15:29.261-0500 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-07-06T11:15:29.264-0500 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2017-07-06T11:15:29.264-0500 I CONTROL [initandlisten]
2017-07-06T11:15:30.821-0500 W FTDC [initandlisten] Failed to initialize Performance Counters for FTDC: WindowsPdhError: PdhExpandCounterPathW failed with 'El objeto especificado no se encontró en el equipo.' for counter '\Memory\Available Bytes'
2017-07-06T11:15:30.822-0500 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory 'c:/Program Files/MongoDB/data/db/diagnostic.data'
2017-07-06T11:15:30.860-0500 I INDEX [initandlisten] build index on: admin.system.version properties: { v: 2, key: { version: 1 }, name: "incompatible_with_version_32", ns: "admin.system.version" }
2017-07-06T11:15:30.860-0500 I INDEX [initandlisten] building index using bulk method; build may temporarily use up to 500 megabytes of RAM
2017-07-06T11:15:30.869-0500 I INDEX [initandlisten] build index done. scanned 0 total records. 0 secs
2017-07-06T11:15:30.871-0500 I COMMAND [initandlisten] setting featureCompatibilityVersion to 3.4
2017-07-06T11:15:30.873-0500 I NETWORK [thread1] waiting for connections on port 27017

```

4. Buat file konfigurasi C:\ Program Files \ MongoDB \ Server \ 3.4 \ bin \ mongod.conf
Dengan isian sebagai berikut:

systemLog:

destination: file

path: C:\ Program Files \ MongoDB \ data \ db \ log \ mongod.log

storage:

dbPath: C:\ Program Files \ MongoDB \ data \ db

Dengan konfigurasi tersebut kita tidak perlu lagi menggunakan parameter --dbpath

5. Melakukan koneksi ke MongoDB

- a. Buka command line window kedua sebagai administrator dan run C:\ Program Files \ MongoDB \ Server \ 3.4 \ bin \ mongo.exe

```

C:\Program Files\MongoDB\Server\3.4\bin>mongo
MongoDB shell version v3.4.6
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.6
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
    http://docs.mongodb.org/
Questions? Try the support group
    http://groups.google.com/group/mongodb-user
Server has startup warnings:
2017-07-06T11:15:29.261-0500 I CONTROL [initandlisten]
2017-07-06T11:15:29.261-0500 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-07-06T11:15:29.264-0500 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2017-07-06T11:15:29.264-0500 I CONTROL [initandlisten]

```

- b. Jalankan perintah (<https://www.mongodb.com/docs/manual/reference/command/>) untuk menampilkan data tabel dibawah ini.

show dbs	
analisisProveedores	

6. Baca manual referensi dan tuliskan fungsi untuk memanggil perintah berikut:

Aggregate	
-----------	--

Count	
Distinct	
Group	
mapReduce	
Find	
Insert	
Update	
Delete	
findAndModify	
parallelCollectionScan	
Logout	
Authenticate	
createUser	
dropUser	
grantRolesToUser	
usersInfo	
renameCollection	
Copydb	
dropDatabase	
listCollections	
drop	
create	
clone	
createIndexes	
dropIndexes	
shutdown	

11.3 Tutorial 2

Tujuan: Mengimpor kumpulan pemasok ke database

1. Jika mongodb tidak tersedia, jalankan mongodb dengan file konfigurasi (lihat Tutorial 1)
2. Jalankan mongoimport binary
 - a. Collection dapat ditemukan di file provider.json. Ingatlah untuk menunjukkan jalur yang sesuai sehingga file tersedia dan biner mongoimport dapat memuatnya.
 - b. perintahnya: mongoimport --db <dbname>--collection <collection name> --drop -file providers.json
3. List database yang ada
4. Buka database provider
5. Show database

11.4 Tutorial 3

Gunakan database yang Anda buat selama praktikum sebelumnya, dan jalankan perintah sesuai dengan requirement:

1. Sisipkan data pada collection providers (contoh sebagai berikut):

```
db.providers.insert( {
  "address" : { "street" : "2 Avenue", "zipcode" : "10075", "building" : "1480",
  "coord" : [ -73.9557413, 40.7720266 ] },
```

- ```
"borough" : "Manhattan",
"cuisine" : "Italian",
"grades" : [{ "date" : ISODate("2014-10-01T00:00:00Z"), "grade" : "A", "score" : 11},
{ "date" : ISODate("2014-01-16T00:00:00Z"), "grade" : "B", "score" : 17}],
"name" : "Vella",
"restaurant_id" : "41704620"
})
```
2. Tampilkan semua dokumen collection providers
  3. Hitung jumlah dokumen yang disimpan dalam collection providers
  4. Buatlah query untuk menampilkan provider restoran dari Manhattan
  5. Buatlah query untuk menampilkan jumlah provider restoran dari Manhattan
  6. Buatlah query untuk menampilkan provider restoran yang memiliki kode pos 10075
  7. Buatlah query untuk menampilkan provider dengan grade B
  8. Dengan menggunakan operator:
    - a. Tampilkan provider yang memiliki score lebih dari 30.
    - b. Tampilkan provider yang memiliki score kurang dari 10.
  9. Buatlah query untuk menampilkan provider dengan Italian cuisine dan zipcode 10075
  10. Buatlah query untuk menampilkan provider secara berurutan berdasarkan borough dan zipcode

## Modul 12. Scripting SQL (Create index, Create view, Analyze Table)

### Tujuan Praktikum

- 1.1 Praktikan mampu mengoperasikan index, view
- 1.2 Praktikan mampu menganalisis table

### 12.1. Ringkasan Materi

#### 12.1.1. Analyze Table

Analyze Table membantu Oracle untuk menentukan berapa banyak baris dalam tabel dan bagaimana penyimpanan dialokasikan. Informasi terpenting yang didapat optimizer dari proses ini adalah jumlah baris dan jumlah blok yang dimiliki Tabel.

##### Manfaat Analyze Table

- 2.1.1.1. Saat menggabungkan dua tabel atau lebih (Join Table), maka kita perlu menganalisis semua tabel yang akan digunakan dalam penggabungan (Join).
- 2.1.1.2. Meningkatkan kinerja. Optimizer akan mencoba menggunakan tabel dengan jumlah baris atau blok paling sedikit sebagai tabel driving.
- 2.1.1.3. Mengurangi jumlah total I/O disk yang diperlukan, dan dengan demikian meningkatkan kinerja.

Saat menganalisis tabel, Oracle mengisi beberapa kolom berikut dalam tampilan kamus data berikut: DBA\_TABLES, ALL\_TABLES, dan USER\_TABLES

Kolom yang diisi adalah:

- 1. NUM\_ROWS : jumlah baris pada tabel.
- 2. BLOK: jumlah blok data yang digunakan.
- 3. EMPTY\_BLOCKS : jumlah blok data.

Note: BLOK + EMPTY\_BLOCKS + 1 sama dengan jumlah total blok yang dialokasikan ke tabel.

#### 12.1.2. Perintah Analyze

Perintah analyze yang digunakan adalah untuk menganalisa suatu tabel, sehingga mendapatkan statistik yang lebih cepat dan lebih baik menggunakan prosedur yang disediakan.

Tabel analisis dapat digunakan untuk membuat statistik untuk sebuah tabel, indeks atau cluster.

Syntax:

```
ANALYZE table tableName {compute|estimate|delete} statistics options
ANALYZE table indexName {compute|estimate|delete} statistics options
ANALYZE cluster clusterName {compute|estimate|delete} statistics options
```

Contoh perintah ANALYZE TABLE:

```
ANALYZE TABLE scott.emp COMPUTE STATISTICS FOR TABLE
```

## 12.2. Studi Kasus

1. Buatlah desain database untuk suatu perusahaan:
  - a. Table department dengan primary key adalah department\_id. Tabel departemen terdiri atas field-field: department\_id, department\_name, manager\_id, dan location\_id.
  - b. Table employee dengan constraint foreign key adalah department\_id. Tabel employee terdiri atas field-field: employee\_id, last\_name NOT NULL, email, salary, commission\_pct, hire\_date NOT NULL>
  - c. Tambahkan tabel-tabel lain yang saling berelasi (seperti: tabel persediaan barang, pemasok barang, dst).
2. Analisis table-table dengan compute statistics
3. Isikan data-data yang bersesuaian dengan tabel-tabel pada nomor 1 (masing-masing 10 record).
4. Analisis table-table dengan compute statistics
5. Buat view empvu80 yang berisi id\_number, name, salary, department\_id dari pegawai yang bekerja di department = 80.
6. Kemudian tampilkan struktur dari view empvu80.
7. Buat non-unique index pada kolom FOREIGN KEY yang ada pada tabel EMPLOYEE.

# Modul 13. Scripting Procedure dan Function

## Tujuan Praktikum

- Praktikan mampu mengoperasikan perintah procedure dan function

## 13.1. Ringkasan Materi

### 13.1.1. Procedure

Procedure dan function adalah sebuah blok PL/SQL yang dapat berdiri sendiri dan disimpan sebagai suatu objek di dalam database untuk melakukan tugas-tugas spesifik tertentu. Hal ini akan membuat kode yang dibuat lebih bersifat modular sehingga mudah untuk di-maintain.

Procedure adalah suatu blok PL/SQL yang menyimpan sekumpulan perintah yang tidak disertai dengan pengembalian nilai. Dengan kata lain, procedure hanya melakukan proses tertentu saja.

Prosedur merupakan subprogram PL/SQL yang berdiri sendiri. Kalau kita punya pekerjaan rutin dan command-commandnya pun itu-itu saja, kita bisa menyimpan comand-command tersebut dan memanggilnya kapan saja kita mau. Itulah filosofi dari prosedur.

User yang membuat prosedur harus punya privilege “create procedure”.

```
SQL> GRANT CREATE PROCEDURE TO nama_user;
```

Syntax untuk membuat procedure :

**CREATE** digunakan untuk membuat procedure yang baru

**REPLACE** digunakan untuk mengganti isi procedure yang telah dibuat.

```
CREATE [OR REPLACE] PROCEDURE nama_procedure
```

```
(parameter_1 tipedata,
 parameter_2 tipedata, ...)
```

```
AS variabel_variabel_lokal
```

```
BEGIN
```

```
 Statemen;
```

```
.....
```

```
END
```

Untuk menjalankan prosedur, jalankan:

- SQL> exec Nama\_Procedure;  
atau SQL> execute PROC\_REFRESH\_MYTAB;
- Di block PL/SQL, tulis saja nama prosedur tersebut  

```
DECLARE
BEGIN
 Nama_Procedure;
END;
```

Contoh:

```
CREATE OR REPLACE PROCEDURE cetak AS
 d varchar(50);
BEGIN
 d:=q'(Coba lagi)';
 DBMS_OUTPUT.PUT_LINE(d);
END;
```

Jalankan dengan perintah SQL>exec cetak;

### 13.1.1.1 Parameter pada Procedure

Parameter pada procedure digunakan sebagai penghubung data antara procedure dengan si pemanggil procedure. Parameter yang terdapat pada procedure dinamakan **Formal Parameter**. Sedangkan parameter yang terdapat pada pemanggil procedure adalah **Actual Parameter**. Tipe parameter pada Procedure ada 3, yaitu:

1. Parameter **IN**, merupakan tipe parameter yang didefinisikan pada aktual parameter untuk kemudian ditangkap oleh formal parameter. Kita tidak perlu menuliskan IN untuk mendefinisikan parameter tersebut, karena parameter IN telah didefinisikan secara DEFAULT oleh Oracle.
2. Parameter **OUT**, merupakan tipe parameter pada procedure yang nilainya dapat digunakan oleh pemanggil procedure.
3. Parameter **IN OUT**, merupakan tipe parameter yang digunakan untuk mengirimkan sebuah nilai ke procedure yang kemudian akan diproses dan dikembalikan kepada si pemanggil procedure.

Contoh IN Parameter:

```
CREATE OR REPLACE PROCEDURE raise_salary
 (id IN employees.employee_id%TYPE,
 percent IN NUMBER) IS
BEGIN
 UPDATE employees
 SET salary = salary * (1 + percent/100)
 WHERE employee_id = id;
END raise_salary;
/
EXECUTE raise_salary(176,10)
```

Contoh OUT Parameter:

```
CREATE OR REPLACE PROCEDURE query_emp
 (id IN employees.employee_id%TYPE,
 name OUT employees.last_name%TYPE,
 salary OUT employees.salary%TYPE) IS
BEGIN
 SELECT last_name, salary INTO name, salary
 FROM employees
 WHERE employee_id = id;
END query_emp;

DECLARE
 emp_name employees.last_name%TYPE;
 emp_sal employees.salary%TYPE;
BEGIN
 query_emp(171, emp_name, emp_sal); ...
END;
```

Note: Sesuaikan urutan nilai parameter antara aktual parameter dengan Formal Parameter baik menggunakan IN atau OUT parameter.

#### **Contoh IN OUT Parameter:**

Pertama kita buat terlebih dahulu Procedure format\_phone yang akan mengubah format karakter.

```
SET SERVEROUTPUT ON
CREATE OR REPLACE PROCEDURE format_phone
(phone_num IN OUT VARCHAR2) IS
BEGIN
 phone_num := '(' || SUBSTR(phone_num,1,3) || ')' || SUBSTR(phone_num,4,3) ||
 '-' || SUBSTR(phone_num,7);
END format_phone;
/
```

Setelah itu kita buat pemanggil procedurnya:

```
DECLARE
phone VARCHAR2(21):='234234ASDA';
BEGIN
format_phone(phone);
DBMS_OUTPUT.PUT_LINE(phone);
END;
```

Perhatikan sebelumnya value dari Phone adalah '234234ASDA'. Lalu setelah kita panggil PROCEDURE format\_phone dengan memasukkan variable phone ke dalam parameternya, maka secara otomatis akan mengambil value tersebut (IN parameter), yang kemudia akan diproses dan dikembalikan lagi nilainya (OUT parameter) kedalam bentuk yang berbeda menjadi '(234)234-ASDA'.

#### **13.1.1.2 Passing Parameter**

Terdapat berbagai macam cara dalam melakukan passing parameter.

##### **1. Positional**

Passing Parameter secara Positional adalah passing dengan menyesuaikan urutan pada formal parameter dengan aktual parameter, seperti pada sub bab 13.1.1.1.

##### **2. Named**

Passing Parameter secara Named merupakan cara menspesifikasikan nama variable formal parameter pada parameter aktual dengan menggunakan bantuan '`=>`'.

Contoh:

```
EXECUTE add_dept(loc=>2400, name=>'EDUCATION')
```

##### **3. Combination**

Combination merupakan kombinasi dari Passing secara Positional dengan Named.

Contoh:

```
EXECUTE add_dept('EDUCATION', loc=>2400)
```

### 13.1.1.3 Procedure dalam Procedure

```
CREATE OR REPLACE PROCEDURE cetak (x IN INTEGER) AS
 J INTEGER
BEGIN
 FOR J IN 1..x LOOP
 DBMS_OUTPUT.PUT_LINE(TO_CHAR(J));
 END LOOP;
END;
/
CREATE OR REPLACE PROCEDURE panggil AS
BEGIN
 cetak(10);
END;
/
EXECUTE panggil;
```

## 13.2. Tutorial Procedure

```
SQL> CREATE OR REPLACE PROCEDURE FIBO AS
2 K INT;
3 I INT;
4 J INT;
5 BEGIN
6 K:=1;
7 I:=1;
8 DBMS_OUTPUT.PUT_LINE(TO_CHAR(K));
9 DBMS_OUTPUT.PUT_LINE(TO_CHAR(I));
10 LOOP;
11 J:=K+I;
12 DBMS_OUTPUT.PUT_LINE(TO_CHAR(J));
13 K:=I;
14 I:=J;
15 EXIT WHEN J>20;
16 END LOOP;
17 END;
18 /
```

```
CREATE TABLE MHS (NAMA VARCHAR2(20), NIM VARCHAR2(8) PRIMARY KEY, KELAS VARCHAR2(5))
```

```
INSERT INTO MHS VALUES ('CHOGI KIM', '10118042', 'RPL01');
INSERT INTO MHS VALUES ('TAEYEON KIM', '10118009', 'RPL01');
INSERT INTO MHS VALUES ('YURI KWON', '10118021', 'RPL01');
```

```
SET SERVEROUTPUT ON;
```

```

CREATE OR REPLACE PROCEDURE CETAK AS
NAMA VARCHAR2(20);
NIM VARCHAR2(8);
KELAS VARCHAR2(5);
BEGIN
SELECT * INTO NAMA, NIM, KELAS FROM MHS WHERE NIM = '10118042';
DBMS_OUTPUT.PUT_LINE('NAMA: ' || NAMA);
DBMS_OUTPUT.PUT_LINE('NIM: ' || NIM);
DBMS_OUTPUT.PUT_LINE ('KELAS: ' || KELAS);
SELECT * INTO NAMA, NPM, KELAS FROM MHS WHERE NPM = '10118009';
DBMS_OUTPUT.PUT_LINE ('NAMA : ' || NAMA);
DBMS_OUTPUT.PUT_LINE ('NPM : ' || NPM);
DBMS_OUTPUT.PUT_LINE ('KELAS : ' || KELAS);
SELECT * INTO NAMA, NPM, KELAS FROM MHS WHERE NPM = '10118021';
DBMS_OUTPUT.PUT_LINE ('NAMA : ' || NAMA);
DBMS_OUTPUT.PUT_LINE ('NPM : ' || NPM);
DBMS_OUTPUT.PUT_LINE ('KELAS : ' || KELAS);
END;
/

```

CREATE TABLE MHS (NAMA VARCHAR2 (20), NIM VARCHAR2 (8) PRIMARY KEY, KELAS VARCHAR2 (5));  
SQL> CREATE OR REPLACE PROCEDURE tambah (nim varchar2, nama varchar2, kelas varchar2)  
AS

```

2 begin
3 execute immediate 'truncate table mhs';
4 INSERT INTO MHS VALUES(npm,nama,kelas);
5 commit;
6 end;
7 /

```

### 13.3. Function

Stored Function merupakan sebuah blok PL/SQL yang dapat mengembalikan sebuah nilai. Stored Function juga dapat disimpan dalam sebuah schema object, sehingga dapat digunakan secara berulang-ulang. Berikut Perbedaan Stored Procedure dengan Stored Function.

| Procedure                                       | Function                                   |
|-------------------------------------------------|--------------------------------------------|
| Execute as a PL/SQL statement                   | Invoke as part of an expression            |
| Do not contain RETURN clause in the header      | Must contain a RETURN clause in the header |
| Can return values (if any) in output parameters | Must return a single value                 |
| Can contain a RETURN statement without a value  | Must contain at least one RETURN statement |

Syntax Function :

```

CREATE [OR REPLACE] FUNCTION function_name
[(parameter1 [mode1] datatype1, ...)]
RETURN datatype IS|AS
[local_variable_declarations; ...]

```

```
BEGIN
 – actions;
RETURN expression;
END [function_name];
```

Contoh:

```
SET SERVEROUTPUT ON
CREATE OR REPLACE FUNCTION Lihat_Gaji
 (ID Employees.employee_id%TYPE)
RETURN NUMBER
IS
 Gaji NUMBER;
BEGIN
 SELECT Salary INTO Gaji FROM Employees
 WHERE employee_id = id;
 RETURN Gaji;
END;
/
EXECUTE DBMS_OUTPUT.PUT_LINE (Lihat_Gaji(100))
```

Fungsi diatas adalah fungsi yang digunakan untuk melihat gaji employee berdasarkan employee\_id.

Selain dengan cara diatas, kita juga dapat menggunakan **SQL statement** untuk mendapatkan nilai dari sebuah function.

Contoh :

```
SELECT employee_id, last_name, Lihat_gaji(employee_id) FROM employees
```

Contoh lain :

```
CREATE OR REPLACE FUNCTION Naik_Gaji (Salary NUMBER)
RETURN NUMBER
IS
 Gaji_Sekarang NUMBER;
BEGIN
 Gaji_Sekarang := Salary + 1000 ;
 RETURN Gaji_Sekarang;
END;
/
SELECT Employee_id, Naik_Gaji(salary) FROM Employees WHERE Department_id = 20;
```

Berikut tempat-tempat yang dapat kita gunakan sebuah function pada SQL Statement:

1. Pada **SELECT** list
2. Pada Clausa **WHERE** atau **HAVING**
3. Pada Clausa **CONNECT BY**, **START WITH**, **ORDER BY**, dan **GROUP BY**
4. Pada **VALUES** clause dalam **INSERT** query
5. Pada **SET** clause dalam **UPDATE** query

### **Drop Function**

Kita dapat menghapus fungsi yang telah kita buat dengan menggunakan perintah DROP.

#### Syntax :

```
DROP FUNCTION nama_function
```

### **Return Value**

Sebuah function akan mengembalikan nilai sesuai dengan tipe data yang ditentukan. Untuk memberikan nilai kedalam function setelah diolah, maka dalam function dikenal dengan RETURN.

Contoh:

```
CREATE OR REPLACE FUNCTION nama_function (parameter1,parameter2,...)
RETURN tipe_data
AS variable1 tipe_data; ...
BEGIN statement;
RETURN nilai_yang_dikembalikan
END;
```

### **Function without Parameter**

```
CREATE OR REPLACE FUNCTION tulis
RETURN VARCHAR2
AS
BEGIN
RETURN 'Hello Hello';
END;
```

atau

```
CREATE OR REPLACE FUNCTION tulis
RETURN VARCHAR2
AS hsl VARCHAR2(20);
BEGIN hsl:= 'Hello Hello';
RETURN hsl;
END;
```

### **Function with Parameter**

```
CREATE OR REPLACE FUNCTION isprime(bil INTEGER)
RETURN BOOLEAN AS PRIMA BOOLEAN:=TRUE;
J INTEGER;
BEGIN
IF bil <= 1 THEN
 PRIMA := FALSE;
END IF;
FOR J IN 2..(bil/2) LOOP
 IF MOD(bil,J) = 0 THEN
 PRIMA:=FALSE;
 EXIT;
 END IF;
END LOOP;
RETURN PRIMA;
END;
```

```

CREATE OR REPLACE FUNCTION pangkat (bil INTEGER, n INTEGER)
RETURN INTEGER AS HASIL INTEGER(10); l INTEGER;
BEGIN
 HASIL := 1;
 FOR l IN 1..n LOOP
 HASIL := HASIL * bil;
 END LOOP;
 RETURN HASIL;
END;
/

DECLARE
 H INTEGER;
BEGIN
 H := pangkat(2, 3);
 DBMS_OUTPUT.PUT_LINE('Hasil = ' || TO_CHAR(H));
END;
/

```

### Nested Function

```

CREATE OR REPLACE FUNCTION kuadrat(X NUMBER)
RETURN NUMBER AS HASIL NUMBER(10);
BEGIN
 HASIL := X*X;
 RETURN HASIL;
END;

CREATE OR REPLACE FUNCTION determinan (a NUMBER, b NUMBER, c NUMBER)
RETURN NUMBER AS D NUMBER(10);
BEGIN
 D := kuadrat(b) - (4*a*c);
 RETURN D;
END;

SET SERVEROUTPUT ON
DECLARE
 D NUMBER (10);
BEGIN
 D := determinan(1, 1, -6);
 DBMS_OUTPUT.PUT_LINE('Nilai Determinan = ' || TO_CHAR(D));
END;

```

#### 13.4. Studi Kasus

Apakah output dari contoh program di bawah ini :

```
Create or replace procedure keliling_lingkaran
as R number(5); K number(10);
Begin
 R := 21;
 K := 2*((22/7)*R);
 Dbms_output.put_line ('Keliling Lingkaran = ' || K);
End;
Execute keliling_lingkaran;
```

## Daftar Pustaka

- [1] Gennick, Jonathan. 1999. *Oracle SQL \* plus*: O'reillymedia.
- [2] Connolly, T., & Begg, C. (2005). Database Systems, A Practical Approach to Design, Implementation, and Management. Fourth Edition. Harlow: Pearson Education Limited.
- [3] Henry F. Korth, Abraham Silberschatz. 2011. Database system concepts 6th Edition. McGraw-Hill
- [4] Hafid Asad, 2012, Entity Relationship Diagram dengan Agregasi [online], url: <http://a114403201005480.blogspot.co.id/2012/04/entity-relationship-diagram-dengan.html>
- [5] \_\_\_\_\_, 2014, Basis Data: ER-Lanjutan [online], url: <http://catatancokelat.blogspot.co.id/2014/02/basis-data-er-lanjutan.html>
- [6] Patrycja Dybka, 2015, *Chen Nation*, url : <https://www.vertabelo.com/blog/technical-articles/chen-erd-notation> diakses pada 15 Juli 2019.
- [7] Silberschatz, Korth, Sudarshan. 2010. Database System Concepts. Seventh Edition. Mc Graw Hill Education.
- [8] Code Purbalingga,2017, Kegiatan Belajar 9 : Tahapan Proses Normalisasi [Online], url: <https://codepurbalingga.blogspot.com/2017/07/kegiatan-belajar-9-tahapan-proses.html> diakses pada 5 Agustus 2019.
- [9] Dimas, 2015, Pengertian dan fungsi Erd beserta Contoh Kasus [Online], <http://sang-pencopet.blogspot.com/2015/03/pengertian-dan-fungsi-erd-beserta.html> diakses pada 5 Agustus 2019
- [10] J. A. Hoffer, V. Ramesh, and H. Topi, *Modern Database Management*. Pearson Education, 2016.

