Data Warehouse Engineering

Teknik Desain, Perencanaan, Deployment, SDM, Biaya, dan Green Architecture untuk Sertifikasi Profesional

Onno W. Purbo
onno@indo.net.id
onno@itts.ac.id
Twitter onnowpurbo

Institut Teknologi Tangerang Selatan
Dinas KOMINFO Tangerang Selatan
OnnoCenter
2026

Lisensi: Karya ini dilisensikan di bawah **Creative Commons Attribution (CC BY) 4.0 International**. Anda bebas menggunakan, membagikan, dan mengadaptasi karya ini untuk tujuan apa pun, termasuk komersial, dengan tetap mencantumkan atribusi yang layak kepada penulis.

Disclaimer: Buku ini disusun untuk tujuan edukasi, dokumentasi, dan refleksi publik berdasarkan sumber-sumber yang dianggap kredibel pada saat penulisan. Penulis telah berupaya menjaga akurasi, namun sangat mungkin masih terdapat kekurangan atau kekeliruan; karena itu, buku ini tidak dimaksudkan sebagai satu-satunya rujukan yang final, dan penulis terbuka terhadap koreksi serta masukan yang membangun

Kata Pengantar

Data telah menjadi salah satu aset paling penting dalam organisasi modern. Hampir semua keputusan strategis hari ini membutuhkan data yang **akurat, terintegrasi, historis, aman, dan mudah dianalisis**. Namun dalam prakteknya, banyak organisasi masih menghadapi masalah klasik: data tersebar di banyak sistem, definisi KPI berbeda antar unit, laporan dibuat manual, kualitas data tidak konsisten, dan proses audit sulit dilakukan karena asal-usul data tidak terdokumentasi dengan baik.

Buku **Data Warehouse Engineering** ini disusun sebagai referensi praktis dan profesional untuk memahami, merancang, membangun, mengamankan, mengoperasikan, dan mengevaluasi *Data Warehouse*. Fokus utama buku ini adalah membantu pembaca memahami *Data Warehouse* bukan hanya sebagai sistem penyimpanan data, tetapi sebagai **infrastruktur keputusan organisasi**. Karena itu, pembahasan tidak berhenti pada tabel dan *dashboard*, tetapi juga mencakup arsitektur, *ETL/ELT*, *dimensional modeling*, kualitas data, metadata, keamanan, deployment, operasi harian, backup, *disaster recovery*, perencanaan SDM, perhitungan biaya, *sustainability*, dan persiapan sertifikasi.

Buku ini ditujukan untuk pembaca profesional, seperti *data engineer*, *data architect*, analis data, pengelola sistem informasi, auditor teknologi, manajer TI, konsultan, pengelola data organisasi, serta peserta sertifikasi *Data Warehouse*. Materi disusun agar dapat dipakai sebagai **bahan belajar, panduan implementasi, dan referensi sertifikasi**. Setiap bab berusaha menjaga keseimbangan antara teori dan praktik, dengan pendekatan sekitar **60% konsep dan 40% *hands-on***.

Pendekatan utama buku ini adalah sederhana tetapi disiplin: mulai dari kebutuhan bisnis, pahami sumber data, rancang model yang benar, bangun *pipeline* yang dapat diaudit, jaga kualitas data, amankan akses, hitung kapasitas dan biaya, lalu siapkan operasi harian. Prinsip ini penting karena *Data Warehouse* yang baik tidak hanya harus cepat, tetapi juga **benar, aman, hemat, terdokumentasi, dan dapat dipercaya**.

Buku ini juga menekankan pentingnya kemandirian teknologi dan pemahaman arsitektural. Pembaca didorong untuk tidak hanya menjadi pengguna alat, tetapi menjadi profesional yang mampu memahami alasan desain, risiko implementasi, konsekuensi biaya, dan dampak operasional dari setiap keputusan teknis. Dalam konteks sertifikasi, kemampuan seperti ini jauh lebih penting daripada sekadar menghafal istilah.

Akhirnya, semoga buku ini dapat membantu organisasi dan profesional data membangun *Data Warehouse* yang lebih matang, lebih efisien, lebih aman, dan lebih bermanfaat bagi pengambilan keputusan. Semoga buku ini juga menjadi bekal bagi pembaca yang ingin meningkatkan kompetensi, menyiapkan sertifikasi, atau memimpin implementasi *Data Warehouse* di lingkungan kerja masing-masing.

Jakarta, Mei 2026
Onno W. Purbo

Daftar Isi

Kata Pengantar **3**
Daftar Isi **4**
BAGIAN I — FONDASI DAN ARSITEKTUR DATA WAREHOUSE **24**
BAB 1. Data Warehouse sebagai Infrastruktur Keputusan Organisasi 24
1.1 Mengapa Organisasi Membutuhkan Data Warehouse 25
Masalah utama: data tersebar di banyak sistem 25
Data operasional sulit dipakai untuk analitik 26
Kebutuhan laporan historis, tren, KPI, dan audit 26
1.2 Definisi Data Warehouse 27
Subject-oriented 27
Integrated 28
Time-variant 28
Non-volatile 28
Data Warehouse sebagai single source of truth 29
1.3 Peran Data Warehouse dalam Organisasi 30
Pelaporan manajemen 30
Analitik bisnis 31
Audit dan compliance 31
Prediksi dan perencanaan 32
Fondasi AI dan machine learning 32
1.4 Contoh Kasus Profesional 33
Kasus 1 — Data Warehouse penjualan nasional 33
Kasus 2 — Data Warehouse keuangan 33
Kasus 3 — Data Warehouse proyek infrastruktur 34
Kasus 4 — Data Warehouse layanan publik 34
1.5 Materi Sertifikasi yang Harus Dikuasai 35
Kompetensi inti 35
Pertanyaan kritis untuk peserta sertifikasi 35
Kesalahan umum yang harus dihindari 36
Hands-on Mini. Membuat Peta Sederhana Sumber Data Organisasi **37**
Tujuan praktik 37
Langkah 1 — Tentukan area bisnis 37
Langkah 2 — Identifikasi sumber data 37
Langkah 3 — Tentukan pemilik data 37
Langkah 4 — Nilai risiko kualitas 37
Template praktik 38
Langkah 5 — Tentukan prioritas integrasi 38
Ringkasan Bab 39
Latihan Sertifikasi 40
A. Pertanyaan Konseptual 40
B. Studi Kasus Singkat 40
C. Praktik 40

BAB 2. Arsitektur Data Warehouse: Dari Sumber Data sampai Dashboard.....41
2.1 Komponen Utama Arsitektur..... 42
A. Source System.....43
B. Staging Area..... 44
C. ETL/ELT..... 45
ETL.....45
ELT..... 45
Fungsi transformasi.....46
D. Data Warehouse Core..... 47
E. Data Mart..... 48
F. Metadata Repository.....49
G. OLAP Layer..... 50
H. BI Dashboard.....51
I. Governance dan Security Layer.....52
2.2 Arsitektur Dasar.....53
A. Basic Data Warehouse.....53
B. Data Warehouse dengan Staging Area..... 54
C. Data Warehouse dengan Data Mart..... 55
D. Enterprise Data Warehouse.....56
E. Virtual Data Warehouse..... 57
2.3 Arsitektur Tiga Lapis.....58
A. Data Layer.....58
B. Application / Processing Layer.....59
C. Presentation Layer.....60
2.4 Cloud Data Warehouse..... 61
A. Pemisahan Compute dan Storage.....61
B. Elastisitas Kapasitas..... 62
C. Virtual Warehouse.....62
D. Skalabilitas Horizontal dan Vertikal.....63
E. Risiko Cloud Lock-in dan Biaya Berjalan..... 63
2.5 Contoh Arsitektur Praktis.....64
A. Contoh Arsitektur Ritel..... 65
B. Contoh Arsitektur Konstruksi.....66
C. Contoh Arsitektur Kampus.....67
Hands-on. Menggambar Arsitektur Data Warehouse Sederhana.....68
Tujuan Praktik.....68
Langkah 1 — Pilih Kasus..... 68
Langkah 2 — Identifikasi Sumber Data.....68
Langkah 3 — Buat Alur Arsitektur..... 69
Langkah 4 — Tentukan Tabel Staging.....69
Langkah 5 — Tentukan Data Warehouse Core.....69
Langkah 6 — Tentukan Data Mart.....70
Langkah 7 — Tentukan Dashboard.....70
Langkah 8 — Tambahkan Governance dan Security.....70

Checklist Arsitektur untuk Sertifikasi..... 71
Kesalahan Umum dalam Arsitektur Data Warehouse..... 71
Ringkasan Bab..... 72
Latihan Sertifikasi..... 73
A. Pertanyaan Konseptual..... 73
B. Studi Kasus..... 73
C. Praktik..... 73
BAB 3. Data Governance, Metadata, dan Kualitas Data..... 75
3.1 Data Governance..... 75
Mengapa governance penting?..... 75
A. Data Owner..... 76
B. Data Steward..... 77
C. Data Custodian..... 77
D. Data Standard..... 78
E. Data Policy..... 78
F. Data Access Control..... 79
3.2 Metadata..... 80
A. Business Metadata..... 80
B. Technical Metadata..... 81
C. Operational Metadata..... 81
D. Data Lineage..... 82
E. Data Dictionary..... 83
3.3 Kualitas Data..... 84
A. Completeness..... 84
B. Accuracy..... 85
C. Consistency..... 85
D. Timeliness..... 86
E. Uniqueness..... 86
F. Validity..... 87
3.4 Checklist Kualitas Data..... 88
Checklist dasar..... 88
Contoh aturan kualitas data..... 88
Hands-on Mini. Membuat Matriks Governance, Metadata, dan Kualitas Data..... 89
Tujuan praktik..... 89
Langkah 1 — Pilih satu domain data..... 89
Langkah 2 — Tentukan peran governance..... 89
Langkah 3 — Buat metadata sederhana..... 89
Langkah 4 — Buat aturan kualitas data..... 90
Langkah 5 — Tentukan tindakan jika data gagal..... 90
Ringkasan Bab..... 90
Latihan Sertifikasi..... 91
A. Pertanyaan Konseptual..... 91
B. Studi Kasus..... 91
C. Praktik..... 91

BAGIAN II — DESAIN, MODELING, ETL/ELT, DAN IMPLEMENTASI.....92
BAB 4. Analisis Kebutuhan dan Perencanaan Data Warehouse.....92
4.1 Menentukan Tujuan Bisnis.....92
A. Laporan Apa yang Dibutuhkan?.....93
B. Keputusan Apa yang Ingin Didukung?.....94
C. Siapa Pengguna Utama?.....94
D. Apa KPI Utama?.....95
E. Seberapa Sering Data Harus Diperbarui?.....96
4.2 Identifikasi Stakeholder.....97
A. Direksi.....97
B. Manajer Unit.....97
C. Analis Bisnis.....98
D. Tim TI / Tim Data.....98
E. Auditor.....99
F. Data Owner.....99
G. Pengguna Dashboard.....100
4.3 Analisis Sumber Data.....101
A. Sistem Transaksi.....101
B. File CSV / Spreadsheet.....102
C. API.....102
D. Log Aplikasi.....103
E. Data Eksternal.....104
F. Data Historis.....104
Template Analisis Sumber Data.....105
4.4 Menentukan Scope Awal.....106
A. Mulai dari 1–2 Domain Data.....106
B. Hindari Membangun Semuanya Sekaligus.....107
C. Gunakan Pendekatan MVP → Pilot → Produksi.....107
1. Minimum Viable Product / MVP.....107
2. Pilot.....108
3. Produksi.....108
D. Kriteria Scope yang Baik.....108
4.5 Dokumen Kebutuhan.....109
A. Business Requirement.....109
B. Data Requirement.....110
C. Security Requirement.....111
D. SLA.....112
E. Data Refresh Requirement.....112
F. Retention Requirement.....113
Hands-on. Template Data Warehouse Requirement Brief.....114
1. Identitas Proyek.....114
2. Ringkasan Kebutuhan Bisnis.....114
3. Daftar Laporan dan Dashboard.....115
4. KPI Utama.....116

5. Analisis Sumber Data..... 116
6. Kebutuhan Keamanan..... 117
7. Kebutuhan Refresh Data..... 117
8. SLA..... 118
9. Retensi Data..... 118
10. Batasan Scope Awal..... 119
11. Risiko Awal..... 119
12. Persetujuan..... 120
Checklist Bab 4 untuk Sertifikasi..... 121
Kesalahan Umum dalam Analisis Kebutuhan..... 121
Ringkasan Bab..... 122
Latihan Sertifikasi..... 123
A. Pertanyaan Konseptual..... 123
B. Studi Kasus..... 123
C. Praktik..... 124
BAB 5. Dimensional Modeling: Fact, Dimension, Star Schema, dan Snowflake Schema... 125
5.1 Konsep Dimensional Modeling..... 126
A. Fact Table..... 127
B. Dimension Table..... 128
C. Measure..... 129
D. Grain..... 129
E. Hierarchy..... 130
F. Surrogate Key..... 131
5.2 Menentukan Grain..... 132
A. Contoh Grain Penjualan..... 132
B. Contoh Grain Invoice..... 133
C. Contoh Grain Proyek..... 133
D. Contoh Grain Energi..... 134
E. Checklist Menentukan Grain..... 134
5.3 Star Schema..... 135
A. Struktur Star Schema..... 135
B. Kelebihan Star Schema..... 136
C. Kekurangan Star Schema..... 137
D. Kapan Memakai Star Schema?..... 137
5.4 Snowflake Schema..... 138
A. Contoh Snowflake Schema..... 138
B. Kelebihan Snowflake Schema..... 139
C. Kekurangan Snowflake Schema..... 140
D. Kapan Memakai Snowflake Schema?..... 140
5.5 Slowly Changing Dimension..... 141
A. SCD Type 0 — Nilai Tidak Diubah..... 141
B. SCD Type 1 — Overwrite..... 142
C. SCD Type 2 — Baris Baru untuk Histori..... 143

D. SCD Type 3 — Kolom Tambahan untuk Histori Terbatas..... 144
E. Ringkasan Pemilihan SCD..... 144
5.6 Contoh Desain: FactSales dan Dimensi Penjualan..... 145
A. Menentukan Grain..... 145
B. Tabel FactSales..... 146
C. DimCustomer..... 147
D. DimProduct..... 148
E. DimDate..... 148
F. DimRegion..... 149
G. DimSalesChannel..... 149
Hands-on. Membuat Star Schema untuk Data Penjualan..... 150
Langkah 1 — Gambar Model..... 150
Langkah 2 — Buat Tabel Dimensi..... 150
DimDate..... 150
DimCustomer..... 150
DimProduct..... 151
DimRegion..... 151
DimSalesChannel..... 151
Langkah 3 — Buat FactSales..... 152
Langkah 4 — Masukkan Contoh Data Dimensi..... 153
Langkah 5 — Masukkan Contoh Data Fact..... 153
Langkah 6 — Query Analitik Dasar..... 153
Penjualan bersih per bulan..... 153
Penjualan per produk..... 154
Penjualan per region..... 154
Margin per kanal penjualan..... 154
Langkah 7 — Validasi Kualitas Model..... 155
Kesalahan Umum dalam Dimensional Modeling..... 155
Checklist Sertifikasi Bab 5..... 156
Ringkasan Bab..... 156
Latihan Sertifikasi..... 157
A. Pertanyaan Konseptual..... 157
B. Studi Kasus..... 157
C. Praktik..... 158
BAB 6. ETL dan ELT: Extract, Transform, Load untuk Profesional..... 159
6.1 Perbedaan ETL dan ELT..... 159
A. ETL: Transformasi Sebelum Masuk Data Warehouse..... 160
B. ELT: Load Dulu, Transformasi di Platform Target..... 161
C. Ringkasan Perbandingan ETL dan ELT..... 161
D. Kapan ETL Lebih Cocok?..... 162
E. Kapan ELT Lebih Cocok?..... 162
6.2 Extract..... 163
A. Full Extract..... 163
B. Incremental Extract..... 164

C. Change Data Capture / CDC..... 165
D. API Extraction..... 166
E. File-Based Ingestion..... 167
6.3 Transform..... 168
A. Cleansing..... 168
B. Standardisasi..... 169
C. Deduplication..... 169
D. Join..... 170
E. Agregasi..... 171
F. Business Rule..... 171
G. Data Masking..... 172
6.4 Load..... 173
A. Initial Load..... 173
B. Incremental Load..... 174
C. Batch Load..... 175
D. Near-Real-Time Load..... 176
E. Error Handling..... 177
6.5 Logging dan Audit ETL..... 178
A. Row Count..... 178
B. Error Count..... 179
C. Start Time dan End Time..... 179
D. Source Checksum..... 180
E. Load Status..... 180
F. Data Reconciliation..... 181
Hands-on. Contoh Alur ETL CSV → Staging → Fact Table Menggunakan SQL / Python..... 182
Struktur File CSV..... 182
Langkah 1 — Buat Tabel Staging..... 182
Langkah 2 — Buat Tabel Error..... 183
Langkah 3 — Buat Tabel Log Proses..... 183
Langkah 4 — Buat Tabel Bersih..... 183
Langkah 5 — Muat CSV ke Staging..... 184
Langkah 6 — Contoh Python Sederhana untuk Load CSV..... 185
Langkah 7 — Transformasi dari Raw ke Clean Staging..... 188
Langkah 8 — Catat Error Data yang Tidak Valid..... 189
Langkah 9 — Load ke Fact Table..... 190
Langkah 10 — Rekonsiliasi..... 191
Checklist Desain Pipeline ETL/ELT..... 192
Kesalahan Umum dalam ETL/ELT..... 193
Ringkasan Bab..... 193
Latihan Sertifikasi..... 194
A. Pertanyaan Konseptual..... 194
B. Studi Kasus..... 194
C. Praktik..... 195

BAB 7. Implementasi Data Warehouse: Step-by-Step dari Nol.....196
7.1 Tahapan Implementasi..... 197
A. Project Charter..... 198
B. Requirement Gathering.....199
C. Data Profiling..... 200
D. Logical Design..... 201
E. Physical Design.....202
F. ETL Development.....203
G. Data Quality Testing.....204
H. Security Setup.....205
I. User Acceptance Testing / UAT.....206
J. Deployment.....207
K. Monitoring..... 208
L. Maintenance.....208
7.2 Contoh Proyek Mini: Data Warehouse Penjualan..... 209
A. Tabel Sumber.....209
customer.csv.....209
product.csv..... 209
branch.csv..... 209
sales.csv.....210
B. Output Data Warehouse..... 210
C. KPI Dashboard.....211
7.3 Struktur Folder Proyek..... 212
Fungsi Folder.....213
7.4 Standar Dokumentasi.....214
A. Data Dictionary.....214
B. ETL Mapping.....214
C. Data Lineage.....215
D. Access Matrix.....215
E. Test Case..... 216
F. Deployment Checklist.....216
G. Operational Manual..... 217
Hands-on. Membangun Mini Data Warehouse dari CSV sampai Dashboard... 218
Langkah 1 — Buat Database dan Skema..... 218
Langkah 2 — Buat Tabel Raw Staging.....218
Langkah 3 — Load CSV ke Raw Staging.....220
Langkah 4 — Buat Tabel Dimensi..... 221
Langkah 5 — Buat Tabel Fakta..... 222
Langkah 6 — Load Dimensi..... 222
Load DimCustomer.....222
Load DimProduct.....223
Load DimBranch.....223
Load DimDate dari Data Penjualan..... 224
Langkah 7 — Load FactSales..... 225

Langkah 8 — Buat Data Mart Penjualan....226
Langkah 9 — Buat Query Dashboard....227
Total Penjualan....227
Penjualan per Bulan....227
Penjualan per Cabang....227
Penjualan per Produk....227
Langkah 10 — Visualisasi di Dashboard....228
Opsi A — Apache Superset....228
Opsi B — Metabase....228
Langkah 11 — Pengujian Data....229
Cek Duplikasi Transaksi....229
Cek Nilai Kosong....229
Rekonsiliasi Total Penjualan....229
Cek Margin Negatif....229
Langkah 12 — Dokumentasi Minimum....230
Checklist Implementasi Bab 7....230
Kesalahan Umum Implementasi....231
Ringkasan Bab....231
Latihan Sertifikasi....232
A. Pertanyaan Konseptual....232
B. Studi Kasus....232
C. Praktik....233
BAGIAN III — DEPLOYMENT, OPERASI, SECURITY, GOVERNANCE, DAN OPTIMASI..... 234
BAB 8. Deployment Data Warehouse: Dari Development ke Production....234
8.1 Environment....234
A. Development....235
B. Testing....236
C. Staging....237
D. Production....238
Ringkasan Perbedaan Environment....238
8.2 Deployment Checklist....239
A. Skema Database Siap....240
B. ETL/ELT Sudah Dites....241
C. Hak Akses Sudah Benar....242
D. Backup Aktif....243
E. Monitoring Aktif....244
F. UAT Disetujui....245
G. Dokumentasi Lengkap....245
8.3 Migrasi Awal....246
A. Initial Load....246
B. Historical Load....247
C. Data Validation....248
D. Business Sign-off....249

8.4 Strategi Cutover..... 250
A. Big Bang.....250
B. Parallel Run.....251
C. Phased Rollout.....252
D. Pilot Unit.....253
Ringkasan Strategi Cutover.....254
8.5 Rollback Plan..... 254
A. Snapshot.....255
B. Backup..... 256
C. Restore Procedure.....257
D. Emergency Contact..... 258
E. Incident Response..... 259
Hands-on. Membuat Checklist Deployment Data Warehouse..... 260
A. Identitas Deployment..... 260
B. Checklist Environment..... 260
C. Checklist Skema Database.....261
D. Checklist ETL/ELT..... 261
E. Checklist Data Migration..... 262
F. Checklist Security..... 262
G. Checklist Backup dan Restore.....263
H. Checklist Monitoring.....263
I. Checklist UAT dan Business Sign-off..... 264
J. Checklist Dokumentasi..... 264
K. Checklist Rollback Plan.....265
Kesalahan Umum Deployment.....266
Ringkasan Bab..... 266
Latihan Sertifikasi..... 267
A. Pertanyaan Konseptual.....267
B. Studi Kasus.....267
C. Praktek.....268
Bab 9. Security, Privacy, dan Compliance Data Warehouse..... 269
9.1 Prinsip Keamanan..... 269
A. Least Privilege.....270
B. Segregation of Duties.....270
C. Role-Based Access Control.....271
D. Encryption at Rest.....272
E. Encryption in Transit.....273
F. Audit Log.....274
G. Data Masking.....275
9.2 Klasifikasi Data.....276
A. Public..... 276
B. Internal..... 277
C. Confidential.....277
D. Restricted.....278

E. Personal Data..... 279
9.3 Access Matrix..... 280
Contoh Access Matrix Dasar..... 280
Access Matrix Lebih Detail..... 281
Aturan Tambahan untuk Access Matrix..... 281
Contoh Kontrol SQL Sederhana..... 282
9.4 Audit Readiness..... 283
A. Siapa Akses Data?..... 284
B. Kapan Data Berubah?..... 285
C. Dari Mana Data Berasal?..... 286
D. Siapa Menyetujui Perubahan?..... 287
E. Apakah Data Sensitif Terlindungi?..... 287
Hands-on. Membuat Matriks Akses dan Klasifikasi Data..... 288
Langkah 1 — Identifikasi Domain Data..... 288
Langkah 2 — Klasifikasikan Data..... 288
Langkah 3 — Tentukan Role..... 289
Langkah 4 — Buat Access Matrix..... 289
Langkah 5 — Tentukan Kontrol Tiap Klasifikasi..... 290
Langkah 6 — Buat Checklist Audit..... 290
Langkah 7 — Contoh Policy Singkat..... 290
Checklist Bab 9 untuk Sertifikasi..... 291
Kesalahan Umum..... 291
Ringkasan Bab..... 292
Latihan Sertifikasi..... 293
A. Pertanyaan Konseptual..... 293
B. Studi Kasus..... 293
C. Praktik..... 293
BAB 10. Performance Tuning dan Capacity Planning..... 294
10.1 Faktor Performa..... 294
A. Volume Data..... 295
B. Jumlah User..... 296
C. Kompleksitas Query..... 297
D. Refresh Frequency..... 298
E. Index..... 299
F. Partitioning..... 300
G. Agregasi..... 301
H. Materialized View..... 302
I. Concurrency..... 303
10.2 Indexing dan Partitioning..... 304
A. Strategi Indexing..... 304
B. Kapan Indeks Tidak Membantu?..... 305
C. Strategi Partitioning..... 306
D. Risiko Partitioning..... 307
10.3 Baseline dan Monitoring..... 308

A. Query Response Time..... 308
B. CPU Utilization..... 309
C. Memory Usage..... 309
D. Storage Growth..... 310
E. I/O..... 310
F. Queue Time..... 311
G. Failed Job..... 311
H. ETL Duration..... 312
10.4 Rumus Kapasitas Dasar..... 313
A. Estimasi Ukuran Fact Table..... 313
B. Pertumbuhan Data..... 313
C. Estimasi Storage Total..... 314
D. Estimasi Concurrency..... 314
E. Estimasi Kebutuhan I/O dan ETL..... 315
10.5 Contoh Perhitungan..... 316
Langkah 1 — Hitung Data Bulanan Tahun Pertama..... 316
Langkah 2 — Hitung Pertumbuhan 5 Tahun..... 316
Langkah 3 — Tambahkan Dimensi dan Agregasi..... 317
Langkah 4 — Hitung Index Overhead..... 317
Langkah 5 — Hitung Raw dan Staging..... 317
Langkah 6 — Hitung Backup..... 317
Langkah 7 — Hitung Total Sebelum Cadangan..... 318
Langkah 8 — Tambahkan Cadangan 25%..... 318
Ringkasan Perhitungan..... 318
Hands-on. Menghitung Kapasitas Storage dan Compute untuk DW 5 Tahun..... 319
A. Input Kapasitas..... 319
B. Rumus..... 319
C. Template Perhitungan 5 Tahun..... 320
D. Template Storage Total..... 320
E. Template Estimasi Compute..... 321
F. Contoh Estimasi Compute..... 322
G. Checklist Optimasi Performa..... 322
Kesalahan Umum..... 323
Ringkasan Bab..... 323
Latihan Sertifikasi..... 324
A. Pertanyaan Konseptual..... 324
B. Studi Kasus..... 324
C. Praktik..... 325
BAB 11. Operasi Harian, Monitoring, Backup, dan Disaster Recovery..... 326
11.1 Operasi Harian..... 326
A. Cek ETL..... 327
B. Cek Data Refresh..... 328
C. Cek Error Log..... 329
D. Cek Storage..... 330

E. Cek Query Lambat....331
F. Cek Akses User....332
G. Cek Backup....333
11.2 Backup....334
A. Full Backup....334
B. Incremental Backup....335
C. Snapshot....335
D. Archive....336
E. Backup Metadata....337
F. Backup Script ETL....337
11.3 Disaster Recovery....338
A. RPO....338
B. RTO....339
C. Secondary Site....340
D. Restore Test....341
E. Failover Drill....343
11.4 Maintenance....344
A. VACUUM / ANALYZE....344
B. Rebuild Index....345
C. Update Statistics....345
D. Partition Maintenance....346
E. Cleanup Staging....347
F. Archive Old Data....348
Hands-on. Template Checklist Operasi Harian Data Warehouse....349
A. Identitas Operasi....349
B. Checklist ETL....349
C. Checklist Data Refresh....350
D. Checklist Error Log....350
E. Checklist Storage....350
F. Checklist Query dan Performa....351
G. Checklist Akses User....351
H. Checklist Backup....351
I. Checklist Maintenance....352
J. Eskalasi....352
K. Catatan Harian....352
Checklist Mingguan....353
Checklist Bulanan....353
Kesalahan Umum Operasi....354
Ringkasan Bab....354
Latihan Sertifikasi....355
A. Pertanyaan Konseptual....355
B. Studi Kasus....355
C. Praktik....355
BAGIAN IV. SDM, BIAYA, SUSTAINABILITY, DAN SERTIFIKASI....356

BAB 12. Perencanaan SDM Data Warehouse....356
12.1 Struktur Tim Minimal....356
A. Sponsor Bisnis....357
B. Product Owner Data....357
C. Data Architect....358
D. Data Engineer....358
E. ETL Developer....359
F. DBA / Platform Engineer....359
G. BI Developer....360
H. Data Analyst....360
I. Data Steward....361
J. Security / Compliance Officer....361
K. Project Manager....362
12.2 Struktur Tim Kecil....363
A. Pola Tim Kecil 3 Orang....363
B. Pola Tim Kecil 5 Orang....364
12.3 Struktur Tim Menengah....365
A. Contoh Struktur Tim Menengah....365
B. Kapan Struktur Menengah Dibutuhkan?....366
C. Risiko Tim Menengah....366
12.4 Struktur Tim Enterprise....367
A. Data Platform Team....367
B. Data Governance Office....368
C. Data Engineering Squad....368
D. BI & Analytics Squad....369
E. Security & Compliance....369
F. FinOps / Cost Controller....370
G. Data Quality Team....370
12.5 Kompetensi Sertifikasi....371
A. Mendesain Arsitektur....371
B. Membaca Kebutuhan Bisnis....372
C. Membuat Model Dimensional....372
D. Merancang ETL/ELT....372
E. Mengelola Metadata....373
F. Menjamin Kualitas Data....373
G. Menghitung Kapasitas....374
H. Mengelola Keamanan....374
I. Menghitung Biaya....374
J. Menyusun Rencana Operasi....375
12.6 Estimasi Biaya SDM....376
A. Contoh Estimasi Biaya SDM Proyek DW....376
B. Tambahan Biaya yang Sering Terlupakan....377
C. Estimasi Berdasarkan Fase....378
D. Estimasi Tim Kecil....378

E. Estimasi Tim Menengah....379
F. Estimasi Tim Enterprise....380
Hands-on. Template Perencanaan SDM dan Biaya Data Warehouse....381
A. Identitas Proyek....381
B. Matriks Peran dan Tanggung Jawab....381
C. RACI Sederhana....382
D. Matriks Kompetensi....383
E. Template Estimasi Biaya SDM....383
F. Template Biaya Tambahan....384
G. Checklist Kesiapan SDM....384
Kesalahan Umum Perencanaan SDM....385
Ringkasan Bab....385
Latihan Sertifikasi....385
A. Pertanyaan Konseptual....386
B. Studi Kasus....386
C. Praktik....387
BAB 13. Perhitungan Biaya Data Warehouse....388
13.1 Komponen Biaya....388
A. Perangkat Keras / Cloud....389
B. Storage....390
C. Compute....391
D. Network....392
E. Backup....392
F. Lisensi Software....393
G. Open Source Support....393
H. SDM Internal....394
I. Konsultan....394
J. Training....395
K. Security....395
L. Monitoring....396
M. Disaster Recovery....396
N. Operasional Bulanan....397
13.2 CAPEX vs OPEX....398
A. CAPEX....398
B. OPEX....399
C. Perbandingan CAPEX dan OPEX....400
13.3 Rumus TCO....401
Komponen Implementasi Awal....401
Komponen Operasional Bulanan....402
Cadangan Risiko....402
13.4 Rumus Biaya Cloud DW....403
A. Biaya Storage....403
B. Biaya Compute....404
C. Biaya Transfer Data....404

D. Biaya Backup....405
E. Biaya Monitoring....405
13.5 Rumus Biaya On-Premise DW....406
A. Server....406
B. Storage....407
C. Jaringan....407
D. Listrik dan Pendingin....408
E. Maintenance....408
13.6 Rumus Biaya per Query....409
Kapan Metrik Ini Berguna?....409
13.7 Rumus Biaya per Laporan....410
Interpretasi....410
13.8 Contoh Simulasi Biaya....411
A. Hitung Pertumbuhan Data 3 Tahun....411
B. Tambahkan Komponen Storage....412
C. Estimasi Compute....413
D. Simulasi Biaya Bulanan....414
E. Biaya Implementasi Awal....414
F. Hitung TCO 3 Tahun....415
G. Biaya per Query....415
H. Biaya per Laporan....416
Hands-on. Membuat Tabel Estimasi Biaya Data Warehouse 3 Tahun....417
A. Input Dasar....417
B. Estimasi Pertumbuhan Data....417
C. Estimasi Storage....418
D. Estimasi Compute....418
E. Estimasi OPEX Bulanan....419
F. Estimasi Implementasi Awal....419
G. Hitung TCO 3 Tahun....420
H. Biaya per Query dan Laporan....420
I. Checklist FinOps DW....421
Kesalahan Umum Perhitungan Biaya....422
Ringkasan Bab....422
Latihan Sertifikasi....423
A. Pertanyaan Konseptual....423
B. Studi Kasus....423
C. Praktik....424
Bab 14. Environmentally Friendly Data Warehouse....425
14.1 Prinsip Green Data Warehouse....425
A. Hemat Data....426
B. Hemat Compute....426
C. Hemat Perpindahan Data....427
14.2 Metrik Keberlanjutan untuk DW....428
A. PUE....428

B. Konsumsi Energi....429
C. Software Carbon Intensity....430
D. Storage Utilization....431
E. Query Efficiency....431
14.3 Strategi Desain DW yang Ramah Lingkungan....432
A. Pilih Grain yang Tepat....432
B. Gunakan Incremental Load....433
C. Partisi Data Historis....433
D. Gunakan Agregasi dan Materialized View....434
E. Terapkan Retensi dan Arsip....434
14.4 Optimasi Operasional Ramah Lingkungan....435
A. Audit Dashboard....435
B. Review ETL Window....436
C. Review Backup....436
D. Review Query Berat....436
14.5 Rumus Dasar Efisiensi DW....437
A. Efisiensi Storage....437
B. Efisiensi Query....437
C. Efisiensi Dashboard....437
D. Efisiensi Retensi....438
Hands-on. Membuat Checklist Environmentally Friendly Data Warehouse....439
A. Identitas Sistem....439
B. Checklist Data dan Storage....439
C. Checklist Compute dan Query....440
D. Checklist Dashboard dan Laporan....440
E. Checklist Keberlanjutan dan Biaya....441
F. Template Rencana Optimasi....441
Contoh Mini Audit Green DW....442
A. Rasio Storage Aktif....442
B. Rasio Dashboard Aktif....442
C. Energi per Query....442
Kesalahan Umum....443
Ringkasan Bab....443
Latihan Sertifikasi....444
A. Pertanyaan Konseptual....444
B. Studi Kasus....444
C. Praktek....445
BAB 15. Roadmap Implementasi dan Persiapan Sertifikasi Data Warehouse....446
15.1 Roadmap Implementasi 90 Hari....446
Hari 1–15: Assessment Kebutuhan dan Sumber Data....447
Hari 16–30: Desain Arsitektur dan Model Data....448
Hari 31–50: ETL/ELT dan Staging....449
Hari 51–65: Data Mart dan Dashboard....450
Hari 66–75: Security, Governance, dan Testing....451

Hari 76–85: UAT dan Deployment....452
Hari 86–90: Dokumentasi, Training, dan Review....453
15.2 Roadmap Belajar Sertifikasi....454
Minggu 1: Konsep DW, OLTP, OLAP....454
Minggu 2: Arsitektur dan ETL/ELT....455
Minggu 3: Dimensional Modeling....456
Minggu 4: Security, Governance, Metadata....457
Minggu 5: Performance Tuning....458
Minggu 6: Cost, SDM, Deployment, Operasi....459
15.3 Kisi-Kisi Ujian....460
A. Topik Ujian....460
B. Bentuk Soal....461
C. Level Kompetensi....461
15.4 Simulasi Studi Kasus....462
Kasus....462
Tugas 1 — Mendesain Arsitektur....463
Tugas 2 — Membuat Star Schema....464
Model Penjualan....464
Model Proyek....464
Tugas 3 — Menentukan ETL/ELT....465
Tugas 4 — Menghitung Kapasitas....466
Tugas 5 — Menghitung Biaya....466
Tugas 6 — Menentukan Tim....467
Tugas 7 — Membuat Checklist Deployment....467
Hands-on. Template Rencana Sertifikasi dan Implementasi....468
A. Roadmap Implementasi 90 Hari....468
B. Roadmap Belajar 6 Minggu....468
C. Template Jawaban Studi Kasus....469
Checklist Kesiapan Sertifikasi....470
Kesalahan Umum Saat Sertifikasi....471
Ringkasan Bab....471
Latihan Sertifikasi Akhir....472
A. Pertanyaan Konseptual....472
B. Studi Kasus Akhir....472
Lampiran A. Template Data Warehouse Project Charter....473
A.1 Identitas Proyek....473
A.2 Tujuan Bisnis....473
A.3 Scope Proyek....474
A.4 Data Source....474
A.5 Target Pengguna....475
A.6 KPI Utama....475
A.7 Risiko Awal....475
A.8 Estimasi Biaya....476
A.9 Tim Proyek....476

A.10 Timeline....477
A.11 Kriteria Sukses....477
A.12 Persetujuan....478
Lampiran B. Template Requirement dan Data Source Assessment....479
B.1 Identitas Assessment....479
B.2 Ringkasan Kebutuhan Data....479
B.3 Tabel Data Source Assessment....480
B.4 Panduan Pengisian Kolom....480
B.5 Contoh Pengisian....481
B.6 Penilaian Kualitas Data....482
B.7 Skor Prioritas Integrasi....482
B.8 Catatan Risiko dan Tindak Lanjut....483
B.9 Keputusan Assessment....483
Lampiran C. Template Dimensional Model....484
C.1 Identitas Model....484
C.2 Nama Fact Table....484
C.3 Grain....485
C.4 Measure....485
C.5 Dimension....486
C.6 Key....487
C.7 SCD Type....487
C.8 Business Rule....488
C.9 Ringkasan Persetujuan Model....489
C.10 Persetujuan....489
Lampiran D. Rumus Perencanaan Kapasitas....491
D.1 Rumus Ukuran Data....491
D.2 Rumus Growth Data....491
D.3 Rumus Storage Total....492
D.4 Rumus Backup....492
D.5 Rumus Retention....493
D.6 Rumus Compute....493
D.7 Rumus Concurrency....494
D.8 Rumus Network Transfer....495
D.9 Rumus Query Cost....496
D.10 Rumus Energy per Query....496
D.11 Template Ringkas Perhitungan Kapasitas....497
D.12 Catatan Praktis....497
Lampiran E. Template Perhitungan Biaya....499
E.1 Identitas Perhitungan....499
E.2 Template Perhitungan Biaya....499
E.3 Rumus Perhitungan....501
E.4 Contoh Pengisian Ringkas....501
E.5 Catatan Asumsi....502
E.6 Checklist Validasi Biaya....502

Lampiran F. Template Perencanaan SDM..... 504
F.1 Identitas Perencanaan SDM.....504
F.2 Template Perencanaan SDM.....504
F.3 Rumus Biaya SDM..... 506
F.4 Contoh Pengisian Ringkas..... 506
F.5 Matriks Peran yang Bisa Dirangkap..... 506
F.6 Checklist Kesiapan SDM.....507
F.7 Catatan Risiko SDM..... 508
Lampiran G. Checklist Deployment dan Operasi.....509
G.1 Identitas Deployment.....509
G.2 Checklist Deployment.....509
G.3 Detail Validasi Minimum..... 510
G.4 Checklist Operasi Hari Pertama..... 510
G.5 Kriteria Go / No-Go.....511
G.6 Persetujuan Deployment..... 511
Lampiran H. 30 Soal Simulasi Sertifikasi Data Warehouse.....513
H.1 Soal Pilihan Ganda.....513
H.2 Soal Studi Kasus Singkat..... 515
H.3 Soal Perhitungan Kapasitas / Biaya..... 515
H.4 Kunci Jawaban Pilihan Ganda.....516
H.5 Kunci Ringkas Perhitungan..... 516
Glossary.....517
Referensi.....524
Tentang Penulis.....527

BAGIAN I — FONDASI DAN ARSITEKTUR DATA WAREHOUSE

BAB 1. Data Warehouse sebagai Infrastruktur Keputusan Organisasi

Organisasi modern tidak kekurangan data. Yang sering terjadi justru sebaliknya: **data terlalu banyak, tersebar, tidak konsisten, dan sulit dipercaya**. Data transaksi ada di sistem operasional. Data keuangan ada di aplikasi akuntansi. Data pelanggan ada di sistem layanan. Data proyek ada di spreadsheet. Data laporan ada di dokumen terpisah. Ketika manajemen meminta satu angka resmi, sering muncul beberapa versi jawaban.

Di sinilah *Data Warehouse* menjadi penting. *Data Warehouse* bukan hanya tempat menyimpan data. Ia adalah **infrastruktur keputusan**. Fungsinya adalah mengumpulkan data dari berbagai sumber, membersihkan, menyatukan, menyusun, dan menyajikannya agar organisasi dapat membuat keputusan berbasis bukti.

Referensi IBM menjelaskan bahwa *data warehouse* mengagregasikan data dari berbagai sumber ke penyimpanan pusat yang dioptimalkan untuk *query* dan analitik, umumnya melalui proses *ETL* atau *ELT* untuk membersihkan, menyiapkan, dan mengorganisasi data bagi *business intelligence* dan analitik. (IBM) Buku referensi yang digunakan juga menekankan bahwa *data warehouse* dipakai untuk analisis, pelaporan, *OLAP*, skema *star*/*snowflake*, data mart, metadata, dan pengambilan keputusan.

Golden Rule: *Data Warehouse* yang baik tidak dimulai dari teknologi, tetapi dari **keputusan penting apa yang harus dibuat organisasi**.

1.1 Mengapa Organisasi Membutuhkan *Data Warehouse*

Organisasi membutuhkan *Data Warehouse* karena data operasional sehari-hari tidak otomatis siap untuk analitik. Sistem transaksi dirancang untuk mencatat aktivitas cepat, seperti input order, pembayaran, stok, absensi, proyek, pengadaan, atau layanan pelanggan. Sistem seperti ini penting, tetapi tidak selalu cocok untuk menjawab pertanyaan strategis.

Contoh pertanyaan strategis:

- Cabang mana yang paling menguntungkan dalam 3 tahun terakhir?
- Produk mana yang tumbuh tetapi marginnya turun?
- Proyek mana yang mulai menunjukkan risiko keterlambatan?
- Berapa biaya operasional per unit layanan?
- Apakah KPI yang dilaporkan konsisten dengan data transaksi?
- Unit mana yang membutuhkan intervensi manajemen?

Pertanyaan seperti ini membutuhkan data historis, data lintas sistem, data yang sudah dibersihkan, dan definisi bisnis yang konsisten.

Masalah utama: data tersebar di banyak sistem

Dalam organisasi profesional, data biasanya tersebar di berbagai tempat:

Area Organisasi	Contoh Data	Masalah Umum
Keuangan	transaksi, anggaran, pembayaran	kode akun tidak konsisten
Operasi	produksi, proyek, layanan	format laporan berbeda
SDM	pegawai, absensi, kompetensi	data tidak terhubung ke kinerja
Penjualan	pelanggan, order, invoice	definisi pelanggan aktif berbeda
Pengadaan	vendor, kontrak, pembelian	data harga tidak mudah dianalisis
Manajemen Risiko	temuan audit, insiden, mitigasi	tidak terhubung ke data operasional

Masalah kritisnya bukan hanya teknis. Masalah utamanya adalah **kepercayaan terhadap data**. Ketika setiap unit memiliki versi data sendiri, rapat manajemen berubah menjadi debat angka, bukan debat strategi.

Data operasional sulit dipakai untuk analitik

Sistem operasional atau *Online Transaction Processing* (*OLTP*) dirancang untuk transaksi harian. Fokusnya adalah mencatat, memperbarui, dan menjaga integritas transaksi. Sebaliknya, *Online Analytical Processing* (*OLAP*) dirancang untuk analisis historis, agregasi, tren, dan pengambilan keputusan. Referensi yang digunakan membedakan bahwa sistem operasional menangani data dinamis dan transaksi harian, sedangkan *Data Warehouse* digunakan untuk analisis dan pelaporan; *OLTP* optimal untuk transaksi, sementara *OLAP* optimal untuk analitik.

Perbedaannya dapat diringkas sebagai berikut:

Aspek	***OLTP***	***OLAP* / *Data Warehouse***
Tujuan	transaksi harian	analisis dan keputusan
Data	data berjalan / terkini	data historis dan terintegrasi
Operasi utama	insert, update, delete	query, agregasi, analisis
Desain	normalisasi	dimensional / analitik
Pengguna	staf operasional, aplikasi	manajemen, analis, auditor
Pertanyaan	“apa transaksi hari ini?”	“apa tren dan penyebabnya?”
Risiko jika salah pakai	sistem transaksi lambat	analisis tidak lengkap

Kebutuhan laporan historis, tren, KPI, dan audit

Manajemen tidak cukup hanya melihat angka hari ini. Manajemen membutuhkan **tren**. Auditor membutuhkan **jejak data**. Pengambil keputusan membutuhkan **angka yang bisa dipertanggungjawabkan**.

Data Warehouse membantu organisasi menjawab:

- **Apa yang terjadi?** melalui laporan historis.
- **Mengapa terjadi?** melalui analisis dimensi.
- **Apa yang mungkin terjadi?** melalui prediksi.
- **Apa yang harus dilakukan?** melalui KPI, rekomendasi, dan prioritas tindakan.

Di sinilah *Data Warehouse* menjadi fondasi *management reporting*, *business intelligence*, audit, dan analitik lanjutan.

1.2 Definisi *Data Warehouse*

Definisi klasik dari Bill Inmon menyebut *Data Warehouse* sebagai kumpulan data yang **subject-oriented, integrated, time-variant, dan non-volatile** untuk mendukung keputusan manajemen. Definisi ini tetap relevan karena menempatkan *Data Warehouse* sebagai sistem pendukung keputusan, bukan sekadar teknologi penyimpanan. (Wikipedia) Referensi buku yang digunakan juga memuat karakteristik utama tersebut: *subject-oriented*, *integrated*, *time-variant*, dan *non-volatile*.

Subject-oriented

Data Warehouse disusun berdasarkan **subjek bisnis**, bukan berdasarkan aplikasi.

Contoh subjek bisnis:

- pelanggan;
- produk;
- proyek;
- vendor;
- penjualan;
- biaya;
- aset;
- risiko;
- layanan.

Artinya, desain *Data Warehouse* harus mengikuti cara organisasi mengambil keputusan. Bukan mengikuti struktur aplikasi semata.

Contoh kritis:

- Sistem transaksi mungkin menyimpan data pelanggan berdasarkan tabel aplikasi.
- *Data Warehouse* harus menyajikan pelanggan sebagai subjek bisnis: profil pelanggan, histori transaksi, lokasi, segmentasi, nilai ekonomi, keluhan, dan risiko.

Integrated

Data Warehouse harus menyatukan data dari berbagai sumber ke dalam format yang konsisten.

Contoh integrasi:

Masalah Integrasi	Contoh	Solusi di *Data Warehouse*
Kode cabang berbeda	JKT01, Jakarta-1, 001	standarisasi kode cabang
Format tanggal berbeda	12/01/2026 vs 2026-01-12	format tanggal tunggal
Nama pelanggan berbeda	PT ABC, ABC Ltd, ABC Indonesia	*master data matching*
Satuan berbeda	kg, ton, unit	konversi satuan
Definisi KPI berbeda	revenue gross vs net	definisi KPI resmi

Tanpa integrasi, dashboard akan terlihat canggih tetapi isinya berbahaya. **Dashboard yang indah dari data yang salah hanya mempercepat keputusan yang salah.**

Time-variant

Data Warehouse menyimpan data historis. Tujuannya bukan hanya mengetahui kondisi sekarang, tetapi memahami perubahan dari waktu ke waktu.

Contoh pertanyaan *time-variant*:

- Bagaimana tren penjualan 36 bulan terakhir?
- Apakah biaya proyek naik setelah perubahan vendor?
- Apakah performa cabang membaik setelah restrukturisasi?
- Apakah risiko keterlambatan meningkat di kuartal tertentu?

Data historis penting untuk tren, prediksi, audit, dan evaluasi kebijakan.

Non-volatile

Data dalam *Data Warehouse* umumnya tidak diubah seperti sistem transaksi. Data baru ditambahkan secara berkala, tetapi data historis dijaga agar tetap dapat diaudit.

Prinsipnya:

- data transaksi lama tidak dihapus sembarangan;
- perubahan penting harus memiliki jejak;
- koreksi data harus terdokumentasi;
- versi historis harus bisa ditelusuri.

Untuk organisasi profesional, sifat *non-volatile* ini penting karena mendukung **auditability** dan **akuntabilitas keputusan**.

Data Warehouse sebagai *single source of truth*

Single source of truth bukan berarti hanya ada satu database di organisasi. Artinya, organisasi memiliki **sumber rujukan resmi** untuk angka strategis.

Contoh:

- angka pendapatan resmi;
- jumlah pelanggan aktif resmi;
- nilai backlog proyek resmi;
- biaya operasional resmi;
- KPI resmi;
- risiko resmi.

Namun perlu diingat: *single source of truth* tidak muncul hanya dengan membeli teknologi. Ia membutuhkan:

- definisi data yang disepakati;
- metadata;
- data owner;
- kualitas data;
- governance;
- proses validasi;
- kontrol akses;
- audit trail.

DAMA-DMBOK menekankan bahwa data management adalah kerangka global untuk membangun, menskalakan, dan mengelola program data agar menghasilkan nilai nyata bagi organisasi. (DAMA International®)

1.3 Peran *Data Warehouse* dalam Organisasi

Data Warehouse berperan sebagai jembatan antara data operasional dan keputusan manajemen. Ia mengubah data mentah menjadi informasi yang dapat dipakai untuk mengarahkan organisasi.

Pelaporan manajemen

Pelaporan manajemen membutuhkan data yang:

- konsisten;
- tepat waktu;
- mudah dibandingkan;
- dapat ditelusuri;
- sesuai definisi KPI;
- bisa dipakai lintas unit.

Contoh laporan:

- laporan pendapatan bulanan;
- laporan biaya proyek;
- laporan kinerja cabang;
- laporan produktivitas pegawai;
- laporan risiko;
- laporan kepatuhan.

Tanpa *Data Warehouse*, laporan sering dibuat manual dari banyak file. Risiko kesalahan meningkat, waktu habis untuk kompilasi, dan analisis menjadi dangkal.

Analitik bisnis

Data Warehouse memungkinkan analitik yang lebih kuat karena data sudah tersusun secara historis dan lintas dimensi.

Contoh dimensi analitik:

- waktu;
- lokasi;
- produk;
- pelanggan;
- proyek;
- vendor;
- unit kerja;
- kanal layanan;
- kategori biaya.

Contoh analisis:

- tren pendapatan per wilayah;
- margin per produk;
- biaya per proyek;
- performa vendor;
- pola keterlambatan;
- kontribusi pelanggan utama;
- risiko berdasarkan tipe kontrak.

Audit dan *compliance*

Audit membutuhkan data yang dapat dibuktikan. *Data Warehouse* membantu menyediakan:

- histori data;
- metadata;
- *data lineage*;
- kontrol akses;
- rekonsiliasi;
- log pemrosesan;
- standar definisi data.

Untuk organisasi yang diaudit, pertanyaan penting bukan hanya “berapa angkanya?”, tetapi juga:

- dari mana angka berasal?
- siapa pemilik datanya?
- kapan data diperbarui?
- aturan transformasinya apa?
- apakah data pernah dikoreksi?
- siapa yang memiliki akses?

Prediksi dan perencanaan

Prediksi yang baik membutuhkan data historis yang bersih. *Data Warehouse* menyediakan fondasi untuk:

- forecasting penjualan;
- prediksi biaya;
- prediksi keterlambatan proyek;
- prediksi kebutuhan stok;
- prediksi churn pelanggan;
- prediksi risiko fraud;
- perencanaan kapasitas.

Buku referensi yang digunakan menempatkan *Data Warehouse* sebagai sistem untuk mengelola data, menjalankan *OLAP query*, dan mendukung analisis berbagai aplikasi.

Fondasi AI dan *machine learning*

AI tidak dapat berdiri di atas data yang kacau. Model AI membutuhkan data yang:

- bersih;
- historis;
- terstruktur;
- konsisten;
- terdokumentasi;
- legal digunakan;
- memiliki konteks bisnis.

Data Warehouse menjadi fondasi penting untuk AI karena menyediakan data yang lebih siap untuk *training*, evaluasi, analitik prediktif, dan *decision intelligence*. Referensi lain yang digunakan juga menekankan integrasi *Data Warehouse* dengan analitik lanjutan dan AI sebagai arah perkembangan modern.

1.4 Contoh Kasus Profesional

Kasus 1 — *Data Warehouse* penjualan nasional

Sebuah perusahaan memiliki banyak cabang. Setiap cabang membuat laporan penjualan sendiri. Format laporan berbeda, kode produk tidak konsisten, dan data pelanggan ganda. Direksi sulit mengetahui produk mana yang benar-benar tumbuh.

Solusi *Data Warehouse*:

- data penjualan dari cabang dikumpulkan;
- kode produk distandarisasi;
- pelanggan dibersihkan dari duplikasi;
- data dimodelkan dalam *star schema*;
- dashboard nasional dibuat;
- KPI penjualan disepakati.

Output strategis:

- produk paling menguntungkan;
- cabang dengan pertumbuhan tertinggi;
- pelanggan bernilai tinggi;
- wilayah dengan penurunan penjualan;
- tren musiman.

Nilai bisnis: keputusan ekspansi dan promosi menjadi berbasis data, bukan intuisi.

Kasus 2 — *Data Warehouse* keuangan

Sebuah organisasi memiliki data anggaran, realisasi, pembayaran, dan vendor di sistem berbeda. Laporan keuangan bulanan lambat karena harus dikonsolidasikan manual.

Solusi *Data Warehouse*:

- integrasi data anggaran dan realisasi;
- mapping kode akun;
- analisis biaya per unit;
- laporan penyimpangan anggaran;
- rekonsiliasi pembayaran;
- audit trail.

Output strategis:

- realisasi anggaran per unit;
- biaya yang melebihi rencana;
- vendor dengan nilai kontrak besar;
- tren biaya operasional;
- potensi efisiensi.

Nilai bisnis: manajemen dapat mengendalikan biaya lebih cepat dan lebih akurat.

Kasus 3 — *Data Warehouse* proyek infrastruktur

Perusahaan proyek memiliki data progres, biaya, material, alat berat, tenaga kerja, dan risiko dalam banyak sistem. Direksi ingin mengetahui proyek mana yang berisiko terlambat.

Solusi *Data Warehouse*:

- integrasi data progres proyek;
- integrasi data biaya;
- data material dan alat berat;
- histori deviasi jadwal;
- dashboard risiko proyek;
- indikator keterlambatan dini.

Output strategis:

- proyek dengan *cost overrun*;
- proyek dengan deviasi progres;
- vendor bermasalah;
- lokasi dengan produktivitas rendah;
- estimasi risiko keterlambatan.

Nilai bisnis: organisasi dapat melakukan intervensi lebih awal sebelum kerugian membesar.

Kasus 4 — *Data Warehouse* layanan publik

Instansi layanan publik memiliki data permohonan, antrian, penyelesaian, pengaduan, dan kepuasan masyarakat. Namun data tersebar dan sulit dipakai untuk evaluasi layanan.

Solusi *Data Warehouse*:

- integrasi data layanan;
- standar kategori layanan;
- analisis waktu penyelesaian;
- dashboard SLA;
- analisis pengaduan;
- pemetaan wilayah layanan.

Output strategis:

- layanan paling lambat;
- wilayah dengan keluhan tinggi;
- unit dengan beban kerja tinggi;
- pola permintaan musiman;
- prioritas perbaikan layanan.

Nilai bisnis: pelayanan publik dapat diukur, diawasi, dan diperbaiki secara sistematis.

1.5 Materi Sertifikasi yang Harus Dikuasai

Bab ini menjadi fondasi untuk sertifikasi. Peserta tidak cukup menghafal definisi. Peserta harus mampu menjelaskan **mengapa, kapan, dan bagaimana** *Data Warehouse* digunakan.

Kompetensi inti

Kompetensi	Yang Harus Dipahami	Bukti Penguasaan
Konsep DW	fungsi, tujuan, karakteristik	mampu menjelaskan DW sebagai sistem keputusan
OLTP vs *OLAP*	transaksi vs analitik	mampu memilih arsitektur sesuai kebutuhan
Data mart	subset DW untuk unit tertentu	mampu memberi contoh data mart
Metadata	data tentang data	mampu membuat kamus data sederhana
ETL/ELT	integrasi data	mampu menjelaskan alur ekstraksi sampai load
Dimensional modeling	fact, dimension, grain	mampu membuat skema awal
Governance	owner, steward, policy	mampu menyusun kontrol data
Performance	query, index, partisi	mampu mengenali bottleneck
Security	akses, masking, audit	mampu membuat matriks akses
Cost planning	SDM, storage, compute, operasi	mampu membuat estimasi biaya awal

Pertanyaan kritis untuk peserta sertifikasi

Peserta sertifikasi harus mampu menjawab pertanyaan berikut:

- Apa masalah bisnis yang diselesaikan oleh *Data Warehouse*?
- Mengapa sistem transaksi tidak cukup untuk analitik?
- Apa perbedaan *OLTP* dan *OLAP*?
- Apa arti *subject-oriented*, *integrated*, *time-variant*, dan *non-volatile*?
- Mengapa metadata penting?
- Apa risiko jika definisi KPI tidak disepakati?
- Kapan organisasi membutuhkan *data mart*?
- Mengapa *Data Warehouse* penting untuk audit?
- Bagaimana *Data Warehouse* mendukung AI?
- Apa biaya dan risiko membangun *Data Warehouse* tanpa governance?

Kesalahan umum yang harus dihindari

Beberapa kegagalan *Data Warehouse* bukan karena teknologinya lemah, tetapi karena desain dan tata kelolanya buruk.

Kesalahan umum:

- membangun DW tanpa kebutuhan bisnis yang jelas;
- terlalu fokus pada software, bukan keputusan;
- tidak menyepakati definisi KPI;
- tidak memiliki data owner;
- mengabaikan kualitas data;
- tidak membuat metadata;
- tidak menghitung biaya operasi;
- tidak menyiapkan keamanan data;
- membuat dashboard sebelum data siap;
- menganggap DW selesai setelah deployment.

Prinsip kritis: *Data Warehouse* yang gagal biasanya bukan karena kurang data, tetapi karena **kurang disiplin desain, governance, dan ownership**.

Hands-on Mini. Membuat Peta Sederhana Sumber Data Organisasi

Tujuan praktik

Latihan ini membantu peserta memahami bahwa desain *Data Warehouse* harus dimulai dari pemetaan sumber data. Sebelum bicara teknologi, organisasi harus tahu:

- data apa yang dimiliki;
- siapa pemiliknya;
- formatnya apa;
- seberapa sering diperbarui;
- seberapa besar risiko kualitasnya;
- data mana yang penting untuk keputusan.

Langkah 1 — Tentukan area bisnis

Pilih satu area organisasi, misalnya:

- penjualan;
- keuangan;
- proyek;
- pengadaan;
- SDM;
- layanan pelanggan;
- layanan publik;
- operasional cabang.

Contoh pilihan: **Data Warehouse Penjualan Nasional**.

Langkah 2 — Identifikasi sumber data

Catat semua sumber data yang relevan. Jangan hanya sistem besar. Spreadsheet, file CSV, laporan manual, dan API juga termasuk sumber data.

Langkah 3 — Tentukan pemilik data

Setiap data harus punya pemilik. Pemilik data bukan selalu tim TI. Biasanya pemilik data adalah unit bisnis.

Contoh:

- data transaksi dimiliki unit penjualan;
- data invoice dimiliki keuangan;
- data pelanggan dimiliki layanan pelanggan;
- data produk dimiliki unit produk;
- data wilayah dimiliki operasional.

Langkah 4 — Nilai risiko kualitas

Gunakan kategori sederhana:

- **Rendah**: data stabil, format konsisten, jarang kosong.
- **Sedang**: ada beberapa format berbeda, perlu validasi.
- **Tinggi**: sering duplikat, banyak kosong, definisi tidak jelas.

Template praktik

Sumber Data	Pemilik	Format	Frekuensi Update	Risiko Kualitas
Transaksi penjualan	Divisi Penjualan	Database / CSV	Harian	Sedang
Data pelanggan	Layanan Pelanggan	Database	Harian	Tinggi
Data produk	Divisi Produk	Spreadsheet	Mingguan	Sedang
Data cabang	Operasional	Spreadsheet	Bulanan	Rendah
Invoice	Keuangan	Database	Harian	Sedang
Target penjualan	Manajemen	Spreadsheet	Bulanan	Tinggi

Langkah 5 — Tentukan prioritas integrasi

Gunakan skor sederhana:

Kriteria	Skor 1	Skor 3	Skor 5
Nilai bisnis	rendah	sedang	tinggi
Kualitas data	buruk	cukup	baik
Ketersediaan data	sulit	sedang	mudah
Risiko jika salah	rendah	sedang	tinggi

Rumus sederhana:

```
Prioritas = Nilai Bisnis + Kualitas Data + Ketersediaan Data
+ Risiko Jika Salah
```

Contoh:

Sumber Data	Nilai Bisnis	Kualitas	Ketersediaan	Risiko Jika Salah	Total
Transaksi penjualan	5	3	5	5	18
Data pelanggan	5	2	4	5	16
Data produk	4	3	4	3	14
Target penjualan	5	2	3	4	14

Interpretasi:

- skor tinggi = masuk prioritas awal;
- skor rendah = bisa masuk fase berikutnya;
- data berisiko tinggi perlu cleansing dan governance lebih kuat.

Ringkasan Bab

Data Warehouse adalah **infrastruktur keputusan organisasi**. Nilainya bukan pada besarnya database, tetapi pada kemampuannya menyatukan data, menjaga histori, memperbaiki kualitas, mendukung analitik, dan menyediakan angka resmi yang dapat dipercaya.

Poin penting:

- **Data operasional tidak otomatis siap untuk analitik.**
- *OLTP* mendukung transaksi; *OLAP* mendukung analisis.
- *Data Warehouse* memiliki karakteristik *subject-oriented*, *integrated*, *time-variant*, dan *non-volatile*.
- *Data Warehouse* mendukung pelaporan, audit, compliance, prediksi, dan AI.
- Sertifikasi Data Warehouse menuntut pemahaman konsep, desain, governance, keamanan, performa, dan biaya.
- Praktik pertama yang wajib dilakukan adalah **memetakan sumber data organisasi**.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Jelaskan mengapa *Data Warehouse* tidak boleh dipahami hanya sebagai database besar.
2. Apa perbedaan utama antara *OLTP* dan *OLAP*?
3. Mengapa sifat *time-variant* penting dalam analisis manajemen?
4. Apa risiko organisasi jika tidak memiliki *single source of truth*?
5. Mengapa metadata penting untuk audit dan compliance?

B. Studi Kasus Singkat

Sebuah perusahaan memiliki laporan penjualan dari 20 cabang. Setiap cabang menggunakan format spreadsheet berbeda. Direksi ingin melihat penjualan nasional, margin produk, dan performa cabang setiap bulan.

Tugas:

- identifikasi minimal 5 sumber data;
- tentukan pemilik data;
- tentukan risiko kualitas;
- usulkan 3 KPI utama;
- jelaskan mengapa kasus ini membutuhkan *Data Warehouse*.

C. Praktik

Buat tabel pemetaan sumber data untuk organisasi Anda sendiri dengan kolom:

Sumber Data	Pemilik	Format	Frekuensi Update	Risiko Kualitas	Prioritas Integrasi

BAB 2. Arsitektur *Data Warehouse*: Dari Sumber Data sampai *Dashboard*

Arsitektur *Data Warehouse* adalah **peta besar** yang menjelaskan bagaimana data bergerak dari sistem asal menuju informasi yang siap dipakai untuk keputusan. Tanpa arsitektur yang jelas, organisasi mudah terjebak pada proyek yang mahal tetapi rapuh: data masuk, tetapi tidak bersih; laporan muncul, tetapi tidak dipercaya; *dashboard* terlihat modern, tetapi tidak memiliki dasar tata kelola.

Data Warehouse yang baik harus menjawab lima pertanyaan inti:

- **Dari mana data berasal?**
- **Di mana data dibersihkan dan divalidasi?**
- **Bagaimana data ditransformasikan?**
- **Di mana data historis dan analitik disimpan?**
- **Bagaimana pengguna mengakses insight secara aman?**

Referensi arsitektur yang digunakan dalam buku ini menempatkan komponen seperti *source systems*, *staging area*, *ETL tools*, *data warehouse database*, *OLAP server*, *data marts*, metadata, dan *access tools* sebagai bagian penting dari ekosistem *Data Warehouse*. IBM juga menjelaskan bahwa *data warehouse* menyiapkan data melalui proses *ETL* atau *ELT*, termasuk *cleansing*, standardisasi, dan deduplikasi agar data siap digunakan untuk analitik dan *business intelligence*. (IBM)

Golden Rule: Arsitektur *Data Warehouse* yang baik bukan yang paling rumit, tetapi yang **paling jelas aliran datanya, paling mudah diaudit, paling aman, dan paling relevan untuk keputusan bisnis**.

2.1 Komponen Utama Arsitektur

Arsitektur *Data Warehouse* dapat dipahami sebagai rantai nilai data. Data bergerak dari sistem asal, diproses, disimpan, disusun, lalu disajikan dalam bentuk laporan, analitik, atau *dashboard*.

Secara umum, komponen utamanya adalah:

```
Source System
      ↓
Staging Area
      ↓
ETL / ELT
      ↓
Data Warehouse Core
      ↓
Data Mart / OLAP Layer
      ↓
BI Dashboard / Report / Analytics
```

Di luar alur utama tersebut, ada dua lapisan yang wajib ada:

- **Metadata repository**
- **Governance dan security layer**

Keduanya bukan tambahan kecil. Keduanya adalah **fondasi kepercayaan**.

A. *Source System*

Source system adalah sistem asal tempat data dibuat. Data di sini biasanya masih bersifat operasional, mentah, dan belum siap untuk analitik.

Contoh *source system*:

- sistem transaksi penjualan;
- sistem keuangan;
- sistem pengadaan;
- sistem proyek;
- sistem SDM;
- sistem layanan pelanggan;
- file CSV;
- spreadsheet;
- API;
- sensor atau perangkat IoT;
- log aplikasi;
- data eksternal.

Masalah umum pada *source system*:

- format data tidak seragam;
- kode unit berbeda antar sistem;
- data duplikat;
- data kosong;
- definisi bisnis tidak konsisten;
- histori tidak lengkap;
- data sulit diakses;
- tidak ada dokumentasi.

Prinsip kritis: jangan langsung percaya pada data sumber. Data sumber harus **dipahami, diprofilkan, diuji, dan divalidasi** sebelum masuk ke *Data Warehouse*.

B. *Staging Area*

Staging area adalah area penampungan sementara. Data dari berbagai sumber dikumpulkan di sini sebelum dibersihkan, distandardisasi, dan dimasukkan ke *Data Warehouse Core*.

Fungsi *staging area*:

- menampung data mentah;
- mengurangi beban ke sistem operasional;
- tempat validasi awal;
- tempat *data profiling*;
- tempat deteksi duplikasi;
- tempat rekonsiliasi jumlah baris;
- tempat menyimpan jejak ekstraksi.

Referensi *Mastering Data Warehousing* menjelaskan *staging area* sebagai ruang sementara untuk mengumpulkan data dari sistem sumber sebelum proses *ETL*, sekaligus membantu konsolidasi data dan menjaga integritas proses integrasi.

Contoh struktur *staging*:

```
stg_sales_raw
stg_customer_raw
stg_product_raw
stg_branch_raw
stg_invoice_raw
stg_project_progress_raw
```

Isi tabel *staging* sebaiknya masih mendekati bentuk asli sumber. Jangan terlalu cepat mengubah data di tahap ini, karena *staging* berguna untuk audit dan pelacakan kesalahan.

C. *ETL/ELT*

ETL adalah singkatan dari *Extract, Transform, Load*. *ELT* adalah *Extract, Load, Transform*. Keduanya adalah proses integrasi data dari sumber ke sistem target seperti *data warehouse* atau *data lake*. IBM menjelaskan bahwa *ETL* dan *ELT* sama-sama memindahkan data mentah dari sumber ke target, tetapi berbeda pada urutan transformasinya. (IBM)

ETL

Pada *ETL*, data diekstrak, ditransformasikan, lalu dimuat ke *Data Warehouse*.

Alur:

```
Extract → Transform → Load
```

Cocok ketika:

- kualitas data sumber buruk;
- aturan transformasi kompleks;
- data sensitif harus di masking sebelum masuk;
- platform target tidak ideal untuk transformasi berat;
- organisasi membutuhkan kontrol ketat sebelum data disimpan.

ELT

Pada *ELT*, data diekstrak, dimuat dulu ke platform target, lalu ditransformasikan di dalam platform tersebut.

Alur:

```
Extract → Load → Transform
```

Cocok ketika:

- platform target kuat untuk komputasi;
- data besar;
- transformasi ingin dilakukan lebih fleksibel;
- organisasi memakai arsitektur modern berbasis *cloud*;
- perlu eksplorasi data mentah sebelum model final.

Fungsi transformasi

Transformasi data mencakup:

- pembersihan data;
- standarisasi format;
- konversi tipe data;
- penggabungan data;
- validasi aturan bisnis;
- deduplikasi;
- agregasi;
- masking data sensitif;
- pembuatan *surrogate key*;
- pemetaan kode lama ke kode standar.

Referensi yang digunakan juga menempatkan *ETL tools* sebagai komponen penting untuk mengambil data dari berbagai sumber, mentransformasikan agar sesuai skema *warehouse*, dan memuat data untuk analisis.

D. Data Warehouse Core

Data Warehouse Core adalah pusat penyimpanan data analitik yang sudah dibersihkan, disatukan, dan disusun untuk kebutuhan analisis.

Karakteristik *Data Warehouse Core*:

- menyimpan data historis;
- menyatukan banyak sumber;
- mendukung *query* analitik;
- memiliki skema yang terkontrol;
- menggunakan definisi data resmi;
- mendukung audit dan pelaporan;
- menjadi dasar *data mart* dan *dashboard*.

Model yang umum:

- *star schema*;
- *snowflake schema*;
- *fact table*;
- *dimension table*;
- *aggregate table*;
- *historical table*.

Contoh struktur inti:

```
dw_fact_sales
dw_fact_invoice
dw_fact_project_progress
dw_dim_customer
dw_dim_product
dw_dim_date
dw_dim_branch
dw_dim_vendor
```

Prinsip desain: *Data Warehouse Core* harus stabil. Jangan setiap permintaan laporan kecil langsung mengubah struktur inti. Gunakan *data mart* atau lapisan semantik untuk kebutuhan khusus.

E. *Data Mart*

Data mart adalah bagian dari *Data Warehouse* yang dirancang untuk kebutuhan unit, fungsi, atau domain tertentu.

Contoh:

- *data mart* penjualan;
- *data mart* keuangan;
- *data mart* proyek;
- *data mart* SDM;
- *data mart* pengadaan;
- *data mart* layanan pelanggan.

Manfaat *data mart*:

- akses data lebih cepat;
- fokus pada kebutuhan unit tertentu;
- mengurangi kompleksitas;
- memudahkan pembuatan *dashboard*;
- meningkatkan performa analitik;
- memudahkan pengaturan hak akses.

Risiko *data mart*:

- definisi KPI bisa berbeda antar unit;
- data bisa menjadi silo baru;
- duplikasi model;
- tata kelola melemah jika tidak dikontrol.

Prinsip kritis: *data mart* boleh berbeda untuk kebutuhan analitik, tetapi **definisi data utama harus tetap konsisten**.

F. *Metadata Repository*

Metadata adalah “data tentang data”. Dalam arsitektur profesional, metadata wajib ada karena membantu pengguna memahami asal, arti, struktur, transformasi, dan pemilik data.

Jenis metadata:

Jenis Metadata	Isi	Contoh
Business metadata	arti bisnis	definisi pelanggan aktif
Technical metadata	struktur teknis	nama tabel, tipe kolom
Operational metadata	proses	waktu ETL, jumlah baris
Lineage metadata	asal-usul data	sumber → staging → warehouse
Security metadata	hak akses	role, permission, masking

Referensi yang digunakan menekankan bahwa *metadata repository* membantu pemahaman data, *data governance*, integrasi, *data lineage*, *impact analysis*, kolaborasi, dan keamanan metadata.

Tanpa metadata, organisasi akan bertanya hal yang sama berulang-ulang:

- angka ini berasal dari mana?
- definisinya apa?
- siapa pemiliknya?
- kapan diperbarui?
- aturan transformasinya apa?
- apakah data ini boleh diekspor?
- apakah data ini mengandung data sensitif?

G. *OLAP Layer*

OLAP layer adalah lapisan analitik yang memungkinkan pengguna melakukan analisis multidimensi.

Kemampuan umum *OLAP*:

- *slice*;
- *dice*;
- *drill-down*;
- *roll-up*;
- agregasi;
- analisis tren;
- perbandingan antar periode;
- analisis lintas dimensi.

Contoh dimensi:

- waktu;
- wilayah;
- produk;
- pelanggan;
- proyek;
- vendor;
- unit kerja;
- kanal layanan.

Contoh analisis:

- penjualan per bulan per wilayah;
- biaya proyek per kontraktor;
- margin per kategori produk;
- keterlambatan proyek per jenis pekerjaan;
- pengaduan layanan per lokasi.

Referensi yang digunakan menjelaskan bahwa *OLAP* mendukung agregasi kompleks di berbagai tingkat granularitas sehingga pengguna dapat melihat data dari berbagai sudut dan tingkat detail.

H. *BI Dashboard*

BI dashboard adalah wajah akhir dari *Data Warehouse*. Pengguna bisnis biasanya tidak melihat tabel, *pipeline*, atau skema. Mereka melihat *dashboard*, laporan, grafik, dan KPI.

Fungsi *dashboard*:

- menyajikan KPI utama;
- memperlihatkan tren;
- memberi peringatan dini;
- mendukung rapat manajemen;
- mempercepat keputusan;
- memudahkan monitoring operasional.

Pilihan *open source* yang relevan:

- *Apache Superset*
- *Metabase*

Apache Superset adalah platform *open source* untuk eksplorasi dan visualisasi data, dengan kemampuan *no-code visualization builder* dan SQL IDE untuk eksplorasi data. (Apache Superset) Metabase adalah platform *open source business intelligence* yang dapat digunakan untuk bertanya terhadap data, membuat visualisasi, dan menghubungkan berbagai sumber data. (Metabase)

Catatan kritis: *dashboard* adalah produk akhir, bukan titik awal. Jangan mulai proyek *Data Warehouse* dari desain grafik. Mulailah dari **keputusan, data, definisi, kualitas, dan arsitektur**.

I. *Governance* dan *Security Layer*

Lapisan *governance* dan keamanan harus melintasi semua komponen arsitektur.

Ruang lingkupnya:

- klasifikasi data;
- data owner;
- data steward;
- kontrol akses;
- audit log;
- enkripsi;
- masking;
- retensi data;
- persetujuan perubahan;
- standar kualitas data;
- *data lineage*;
- pemantauan penggunaan data.

Dalam referensi arsitektur *cloud data warehouse*, aspek seperti *Role-Based Access Control* atau RBAC, enkripsi, metadata, governance, data quality, disaster recovery, dan cost management ditekankan sebagai elemen penting untuk menjaga keamanan, kepatuhan, dan keberlanjutan platform data.

Prinsip kritis: data yang mudah diakses tetapi tidak aman adalah risiko. Data yang aman tetapi tidak bisa digunakan juga gagal. Arsitektur harus menyeimbangkan **akses, keamanan, kepatuhan, dan produktivitas**.

2.2 Arsitektur Dasar

Tidak semua organisasi membutuhkan arsitektur besar sejak awal. Arsitektur harus mengikuti kebutuhan bisnis, volume data, risiko, anggaran, dan kesiapan SDM.

A. *Basic Data Warehouse*

Arsitektur paling sederhana:

```
Source System → ETL → Data Warehouse → Report/Dashboard
```

Cocok untuk:

- organisasi kecil;
- sumber data terbatas;
- laporan sederhana;
- volume data tidak terlalu besar;
- tahap awal sertifikasi atau pembelajaran.

Kelebihan:

- cepat dibuat;
- mudah dipahami;
- biaya rendah;
- cocok untuk *proof of concept*.

Kekurangan:

- validasi data terbatas;
- audit lebih sulit;
- transformasi bisa bercampur dengan proses load;
- kurang cocok untuk banyak sumber data.

B. *Data Warehouse* dengan *Staging Area*

Arsitektur:

```
Source System → Staging Area → ETL/ELT → Data Warehouse → Dashboard
```

Cocok untuk:

- banyak sumber data;
- data sering kotor;
- perlu rekonsiliasi;
- perlu audit ekstraksi;
- transformasi cukup kompleks.

Kelebihan:

- lebih aman untuk sistem sumber;
- data mentah dapat ditelusuri;
- kesalahan transformasi lebih mudah dicari;
- kualitas data lebih terkontrol.

Kekurangan:

- butuh storage tambahan;
- proses lebih panjang;
- perlu disiplin pengelolaan data sementara.

C. *Data Warehouse* dengan *Data Mart*

Arsitektur:

```
Source System
      ↓
Staging Area
      ↓
Data Warehouse Core
      ↓
Data Mart Penjualan / Keuangan / Proyek / SDM
      ↓
Dashboard Unit
```

Cocok untuk:

- organisasi menengah;
- banyak unit bisnis;
- kebutuhan analitik berbeda;
- manajemen akses per unit;
- performa dashboard perlu ditingkatkan.

Kelebihan:

- analitik lebih fokus;
- query lebih cepat;
- dashboard lebih relevan;
- hak akses lebih mudah dikelola.

Kekurangan:

- potensi duplikasi;
- risiko definisi KPI berbeda;
- butuh governance yang lebih kuat.

D. *Enterprise Data Warehouse*

Enterprise Data Warehouse adalah arsitektur terpusat untuk seluruh organisasi.

Ciri utama:

- mencakup banyak domain data;
- memiliki data model enterprise;
- metadata dan governance kuat;
- mendukung banyak unit;
- terintegrasi dengan banyak sistem;
- memiliki standar keamanan dan operasi.

Cocok untuk:

- perusahaan besar;
- instansi publik;
- lembaga keuangan;
- organisasi multi-cabang;
- organisasi dengan kebutuhan audit tinggi;
- organisasi dengan strategi AI dan analitik jangka panjang.

Kelebihan:

- sumber data strategis terpusat;
- konsistensi KPI lebih kuat;
- mendukung audit;
- mendukung analitik lintas unit;
- dapat menjadi fondasi AI organisasi.

Kekurangan:

- biaya lebih besar;
- implementasi lebih lama;
- membutuhkan SDM matang;
- butuh sponsor manajemen;
- risiko gagal jika scope terlalu luas.

E. *Virtual Data Warehouse*

Virtual Data Warehouse menyediakan akses terpadu ke banyak sumber data tanpa selalu memindahkan semua data secara fisik ke satu repository.

Cocok untuk:

- data tersebar;
- kebutuhan akses cepat;
- organisasi hasil merger;
- data besar yang mahal dipindahkan;
- kebutuhan integrasi sementara.

Kelebihan:

- cepat menyediakan tampilan terpadu;
- mengurangi duplikasi data;
- fleksibel;
- cocok untuk eksplorasi.

Kekurangan:

- performa tergantung sumber;
- kontrol kualitas lebih sulit;
- lineage dan governance lebih kompleks;
- tidak selalu cocok untuk beban analitik berat.

Prinsip kritis: virtualisasi data berguna, tetapi jangan dijadikan alasan untuk menghindari disiplin kualitas data.

2.3 Arsitektur Tiga Lapis

Arsitektur tiga lapis membantu profesional memahami pemisahan tanggung jawab dalam *Data Warehouse*.

1. Data Layer
2. Application / Processing Layer
3. Presentation Layer

Referensi *Data Mining and Warehousing Techniques with Machine Learning Concepts* menempatkan *three-tier data warehouse architecture*, *back-end tools*, metadata repository, dan evaluasi tool sebagai bagian penting dalam arsitektur *Data Warehouse*.

A. Data Layer

Data layer adalah lapisan tempat data disimpan.

Isinya:

- source database;
- file sumber;
- *staging table*;
- *warehouse table*;
- *fact table*;
- *dimension table*;
- *aggregate table*;
- arsip data;
- metadata teknis.

Fokus desain:

- struktur data;
- histori;
- partisi;
- indeks;
- retensi;
- backup;
- kualitas data;
- keamanan penyimpanan.

Contoh:

```
raw_sales
stg_sales
dw_fact_sales
dw_dim_customer
dm_sales_monthly
```

B. *Application / Processing Layer*

Lapisan ini memproses data.

Isinya:

- *ETL/ELT pipeline*;
- validasi data;
- transformasi;
- cleansing;
- deduplikasi;
- agregasi;
- scheduler;
- workflow;
- monitoring;
- error handling.

Contoh proses:

```
extract_customer_data()
validate_customer_id()
standardize_date_format()
remove_duplicate_customer()
load_dim_customer()
```

Fokus desain:

- keandalan proses;
- logging;
- audit;
- otomatisasi;
- pemulihan error;
- performa transformasi;
- dokumentasi aturan bisnis.

Prinsip kritis: proses data harus dapat diulang. Jika transformasi hanya dipahami oleh satu orang, arsitektur belum profesional.

C. *Presentation Layer*

Lapisan ini adalah tempat pengguna mengakses informasi.

Isinya:

- *dashboard*;
- laporan;
- *ad hoc query*;
- visualisasi;
- data export;
- API analitik;
- semantic layer.

Fokus desain:

- kemudahan penggunaan;
- keamanan akses;
- definisi KPI;
- performa query;
- konsistensi laporan;
- pengalaman pengguna.

Contoh tampilan:

- dashboard direksi;
- dashboard penjualan;
- dashboard risiko proyek;
- laporan audit;
- laporan SLA;
- peta performa wilayah.

Prinsip kritis: pengguna tidak peduli seberapa rumit pipeline di belakang. Mereka peduli apakah angka **benar, cepat, jelas, dan bisa dipercaya**.

2.4 *Cloud Data Warehouse*

Cloud Data Warehouse memperkenalkan cara baru dalam merancang kapasitas. Pada arsitektur tradisional, storage dan compute sering terikat erat. Pada arsitektur modern, keduanya dapat dipisahkan.

Snowflake documentation menjelaskan bahwa *virtual warehouse* adalah klaster sumber daya komputasi yang menyediakan CPU, memori, dan temporary storage untuk menjalankan operasi dalam sesi Snowflake. (Dokumentasi Snowflake) Dokumentasi yang sama juga menjelaskan bahwa Snowflake memisahkan storage dan compute sehingga pengelolaan serta penskalaan sumber daya menjadi lebih fleksibel. (Dokumentasi Snowflake) Buku Snowflake yang digunakan juga menekankan pemisahan komputasi dan penyimpanan, *virtual warehouse*, skalabilitas, RBAC, governance, security, deployment region, disaster recovery, serta optimasi.

A. Pemisahan *Compute* dan *Storage*

Pemisahan ini berarti:

- storage menyimpan data;
- compute menjalankan query dan proses;
- compute dapat dinaikkan atau diturunkan sesuai kebutuhan;
- beberapa workload dapat dipisah;
- biaya dapat dikontrol berdasarkan pemakaian.

Contoh pemisahan workload:

Workload	Compute Terpisah	Tujuan
ETL malam	compute besar sementara	mempercepat load
Dashboard harian	compute sedang	stabil untuk pengguna
Ad hoc analysis	compute terpisah	tidak mengganggu dashboard
Data science	compute khusus	eksplorasi lebih bebas

B. Elastisitas Kapasitas

Elastisitas berarti kapasitas dapat dinaikkan atau diturunkan sesuai beban.

Manfaat:

- tidak perlu selalu membeli kapasitas puncak;
- cocok untuk beban musiman;
- dapat memisahkan workload berat;
- lebih fleksibel untuk proyek baru;
- mendukung eksperimen analitik.

Risiko:

- biaya membengkak jika tidak dimonitor;
- query buruk tetap mahal;
- pengguna bisa memakai compute berlebihan;
- perlu kebijakan *auto suspend*, limit, dan monitoring.

Snowflake documentation menyebut warehouse dapat dimulai, dihentikan, dan diubah ukurannya kapan saja untuk menyesuaikan kebutuhan compute. (Dokumentasi Snowflake)

C. *Virtual Warehouse*

Virtual warehouse adalah unit compute untuk menjalankan query atau proses data.

Prinsip desain:

- pisahkan workload penting;
- gunakan ukuran sesuai kebutuhan;
- aktifkan penghentian otomatis saat tidak dipakai;
- monitor biaya per workload;
- hindari satu compute besar untuk semua kebutuhan;
- gunakan jadwal untuk proses berat.

Contoh desain:

```
WH_ETL_DAILY      → untuk pipeline harian
WH_BI_DASHBOARD   → untuk dashboard eksekutif
WH_ANALYST        → untuk query analis
WH_SANDBOX        → untuk eksperimen
```

Prinsip kritis: elastisitas tanpa kontrol biaya adalah jebakan. *Cloud Data Warehouse* harus didampingi disiplin *cost governance*.

D. Skalabilitas Horizontal dan Vertikal

Skalabilitas vertikal berarti menaikkan ukuran compute.
Skalabilitas horizontal berarti menambah klaster atau unit compute untuk menangani beban paralel.

Contoh:

- query lambat karena volume besar → naikkan ukuran compute;
- banyak pengguna bersamaan → tambah compute paralel;
- ETL mengganggu dashboard → pisahkan workload;
- query tertentu sering dipakai → buat agregasi atau materialized view.

Namun, scaling bukan obat semua masalah. Query buruk, model data buruk, dan governance buruk tetap akan menghasilkan biaya tinggi.

E. Risiko *Cloud Lock-in* dan Biaya Berjalan

Cloud Data Warehouse kuat, tetapi tidak bebas risiko.

Risiko utama:

- ketergantungan pada satu platform;
- biaya compute tidak terkendali;
- biaya storage historis membesar;
- biaya transfer data;
- kesulitan migrasi;
- konfigurasi keamanan tidak matang;
- kurangnya SDM yang memahami platform.

Mitigasi:

- gunakan standar SQL sebisa mungkin;
- dokumentasikan transformasi;
- pisahkan data mentah, data bersih, dan data mart;
- jaga metadata;
- pantau biaya per workload;
- buat kebijakan retensi;
- audit akses dan query berat;
- hindari fitur sangat spesifik jika tidak perlu.

Prinsip kritis: jangan memilih *cloud* hanya karena modern. Pilih karena memang sesuai kebutuhan bisnis, kapasitas SDM, risiko, regulasi, dan biaya total.

2.5 Contoh Arsitektur Praktis

Bagian ini menggunakan pendekatan *open source* dan mudah direplikasi untuk pembelajaran, sertifikasi, dan prototipe organisasi.

Contoh desain:

```
PostgreSQL / MariaDB
        ↓
Python / SQL ETL
        ↓
Staging Database
        ↓
Data Warehouse Relasional
        ↓
Data Mart
        ↓
Metabase / Apache Superset
        ↓
Dashboard Manajemen
```

Object storage dapat dipakai untuk arsip data mentah, file history, backup, dan data semi-terstruktur.

A. Contoh Arsitektur Ritel

Sumber data:

- transaksi penjualan;
- data pelanggan;
- data produk;
- data cabang;
- data stok;
- target penjualan.

Model:

```
FactSales
DimCustomer
DimProduct
DimDate
DimBranch
DimSalesChannel
```

Dashboard:

- penjualan harian;
- margin per produk;
- performa cabang;
- pelanggan aktif;
- tren penjualan;
- stok kritis.

B. Contoh Arsitektur Konstruksi

Sumber data:

- data proyek;
- progres mingguan;
- biaya proyek;
- data vendor;
- material;
- alat berat;
- tenaga kerja;
- insiden QHSE.

Model:

```
FactProjectProgress
FactProjectCost
FactQHSEIncident
DimProject
DimVendor
DimDate
DimLocation
DimWorkPackage
```

Dashboard:

- progres proyek;
- deviasi jadwal;
- deviasi biaya;
- performa vendor;
- risiko keterlambatan;
- insiden QHSE;
- produktivitas proyek.

C. Contoh Arsitektur Kampus

Sumber data:

- data mahasiswa;
- data dosen;
- data mata kuliah;
- nilai;
- kehadiran;
- pembayaran;
- kelulusan.

Model:

```
FactEnrollment
FactGrade
FactAttendance
FactPayment
DimStudent
DimLecturer
DimCourse
DimSemester
DimProgram
```

Dashboard:

- jumlah mahasiswa aktif;
- kelulusan per program;
- performa akademik;
- tingkat kehadiran;
- tunggakan pembayaran;
- tren penerimaan mahasiswa.

Hands-on. Menggambar Arsitektur *Data Warehouse* Sederhana

Tujuan Praktik

Peserta mampu menggambar arsitektur *Data Warehouse* sederhana untuk organisasi ritel, konstruksi, atau kampus. Fokus latihan bukan pada alat gambar, tetapi pada **logika aliran data**.

Langkah 1 — Pilih Kasus

Pilih satu:

- perusahaan ritel;
- perusahaan konstruksi;
- kampus;
- layanan publik;
- rumah sakit;
- koperasi;
- perusahaan distribusi.

Contoh yang digunakan di sini: **perusahaan konstruksi**.

Langkah 2 — Identifikasi Sumber Data

Gunakan tabel berikut:

Sumber Data	Pemilik	Format	Frekuensi Update	Risiko Kualitas
Data proyek	PMO	Database	Harian	Sedang
Progres mingguan	Tim proyek	Spreadsheet	Mingguan	Tinggi
Biaya proyek	Keuangan	Database	Harian	Sedang
Vendor	Pengadaan	Spreadsheet	Bulanan	Sedang
Material	Logistik	CSV / Database	Harian	Sedang
Insiden QHSE	QHSE	Form / CSV	Harian	Tinggi

Langkah 3 — Buat Alur Arsitektur

Gunakan gambar teks sederhana:

```
[Data Proyek]          \
[Progres Mingguan]      \
[Biaya Proyek]           → [Staging Area] → [ETL/ELT] → [Data Warehouse Core]
[Vendor]                /                                      ↓
[Material]             /                                [Data Mart Proyek]
[Insiden QHSE]         /                                     ↓
                                                      [Dashboard Manajemen]
```

Langkah 4 — Tentukan Tabel *Staging*

Contoh:

Nama Tabel Staging	Sumber	Fungsi
stg_project	data proyek	menampung data proyek mentah
stg_progress	progres mingguan	validasi progres
stg_cost	biaya proyek	rekonsiliasi biaya
stg_vendor	vendor	standardisasi nama vendor
stg_material	material	validasi satuan
stg_qhse	insiden QHSE	klasifikasi insiden

Langkah 5 — Tentukan Data Warehouse Core

Contoh:

Tabel	Jenis	Isi
fact_project_progress	Fact	progres proyek per periode
fact_project_cost	Fact	biaya proyek
fact_qhse_incident	Fact	insiden QHSE
dim_project	Dimension	profil proyek
dim_vendor	Dimension	profil vendor
dim_date	Dimension	kalender
dim_location	Dimension	lokasi proyek
dim_work_package	Dimension	paket pekerjaan

Langkah 6 — Tentukan *Data Mart*

Contoh:

Data Mart	Pengguna	Tujuan
dm_project_dashboard	Direksi / PMO	monitoring proyek
dm_cost_control	Keuangan	kontrol biaya
dm_qhse_monitoring	QHSE	keselamatan dan kepatuhan
dm_vendor_performance	Pengadaan	evaluasi vendor

Langkah 7 — Tentukan Dashboard

Contoh *dashboard*:

- progres proyek aktual vs rencana;
- deviasi biaya;
- proyek berisiko tinggi;
- insiden QHSE per lokasi;
- performa vendor;
- tren keterlambatan;
- top 10 proyek dengan risiko biaya.

Langkah 8 — Tambahkan Governance dan Security

Tambahkan kontrol:

Area	Kontrol
Akses data proyek	role berdasarkan unit
Data biaya	hanya untuk keuangan dan manajemen
Data QHSE	akses terbatas
Metadata	definisi KPI dan data lineage
Audit	log akses dashboard dan query
Retensi	data proyek disimpan minimal sesuai kebijakan organisasi
Backup	backup harian dan uji restore berkala

Checklist Arsitektur untuk Sertifikasi

Gunakan checklist ini untuk mengevaluasi desain arsitektur:

Pertanyaan	Ya/Tidak
Apakah semua source system sudah dipetakan?	
Apakah ada staging area?	
Apakah proses ETL/ELT terdokumentasi?	
Apakah ada data warehouse core?	
Apakah data mart dibuat berdasarkan kebutuhan bisnis?	
Apakah metadata tersedia?	
Apakah ada definisi KPI resmi?	
Apakah ada kontrol akses berbasis peran?	
Apakah ada audit log?	
Apakah dashboard mengambil data dari sumber yang benar?	
Apakah risiko biaya sudah dipertimbangkan?	
Apakah ada strategi backup dan pemulihan?	

Kesalahan Umum dalam Arsitektur *Data Warehouse*

Kesalahan yang sering terjadi:

- langsung membuat *dashboard* tanpa desain data;
- tidak memiliki *staging area*;
- semua transformasi ditaruh dalam laporan;
- tidak ada metadata;
- tidak ada pemilik data;
- tidak ada standar penamaan tabel;
- tidak ada kontrol akses;
- tidak memisahkan workload ETL dan dashboard;
- tidak menghitung biaya operasi;
- tidak memiliki strategi backup;
- membiarkan setiap unit membuat definisi KPI sendiri.

Prinsip kritis: arsitektur buruk akan membuat proyek *Data Warehouse* terlihat berhasil di awal, tetapi mahal, lambat, dan sulit dipercaya saat dipakai serius.

Ringkasan Bab

Data Warehouse adalah ekosistem, bukan satu database tunggal. Arsitekturnya harus menjelaskan aliran data dari *source system* sampai *dashboard*. Komponen pentingnya meliputi *source system*, *staging area*, *ETL/ELT*, *Data Warehouse Core*, *data mart*, metadata, *OLAP layer*, *BI dashboard*, serta lapisan *governance* dan keamanan.

Poin penting:

- **Arsitektur harus dimulai dari aliran data, bukan dari tampilan dashboard.**
- *Staging area* penting untuk validasi, audit, dan integrasi.
- *ETL/ELT* menentukan kualitas dan keandalan data.
- *Data Warehouse Core* adalah pusat data historis dan analitik.
- *Data mart* mempercepat analitik unit, tetapi harus dikontrol governance.
- Metadata adalah fondasi pemahaman, audit, dan *data lineage*.
- *Cloud Data Warehouse* memberi elastisitas, tetapi membawa risiko biaya dan *lock-in*.
- *Dashboard* hanya bernilai jika datanya benar, aman, dan dapat dipercaya.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Jelaskan fungsi *staging area* dalam arsitektur *Data Warehouse*.
2. Apa perbedaan utama antara *Data Warehouse Core* dan *data mart*?
3. Mengapa metadata penting dalam arsitektur profesional?
4. Jelaskan perbedaan *ETL* dan *ELT*.
5. Apa risiko utama *Cloud Data Warehouse* dari sisi biaya dan ketergantungan platform?

B. Studi Kasus

Sebuah perusahaan konstruksi ingin membuat dashboard proyek nasional. Data berasal dari spreadsheet progres mingguan, sistem keuangan, data vendor, data material, dan laporan QHSE.

Tugas:

- gambar arsitektur sederhana;
- tentukan tabel *staging*;
- tentukan minimal 3 *fact table*;
- tentukan minimal 5 *dimension table*;
- tentukan 3 *data mart*;
- tentukan kontrol keamanan.

C. Praktik

Buat gambar arsitektur *Data Warehouse* untuk organisasi Anda sendiri dengan struktur:

```
Source System → Staging Area → ETL/ELT → Data Warehouse Core
→ Data Mart → Dashboard
```

Tambahkan:

- metadata;
- governance;
- security;
- backup;
- monitoring;
- risiko kualitas data.

BAB 3. Data Governance, Metadata, dan Kualitas Data

Data Warehouse tidak otomatis menghasilkan keputusan yang benar. Ia hanya mempercepat akses ke data. Jika data yang masuk salah, tidak lengkap, tidak konsisten, tidak punya pemilik, dan tidak punya definisi resmi, maka hasil akhirnya tetap berbahaya: **laporan cepat, tetapi menyesatkan**.

Masalah terbesar dalam *Data Warehouse* bukan hanya kapasitas, *query*, atau *dashboard*. Masalah terbesarnya adalah **kepercayaan terhadap data**. Data yang tidak dipercaya akan ditinggalkan oleh pengguna. Data yang salah tetapi dipercaya akan merusak keputusan.

Karena itu, *data governance*, metadata, dan kualitas data adalah fondasi profesional. DAMA-DMBOK diposisikan sebagai referensi global untuk praktik manajemen data yang efektif dan terstandar lintas industri. DAMA juga menjelaskan bahwa data management mencakup beberapa area pengetahuan penting seperti *data governance*, *data quality*, *metadata*, *data security*, *data warehousing*, dan *business intelligence*. (DAMA International®)

Golden Rule: *Data Warehouse* yang tidak punya *governance* bukan aset strategis, tetapi **risiko organisasi yang diberi tampilan dashboard**.

3.1 *Data Governance*

Data governance adalah tata kelola agar data memiliki **pemilik, aturan, standar, kualitas, keamanan, dan akuntabilitas**. Dalam konteks *Data Warehouse*, *governance* memastikan data yang masuk, diproses, disimpan, dan disajikan dapat dipercaya.

Referensi *Mastering Data Warehousing* menjelaskan bahwa *data governance* mencakup pengelolaan ketersediaan, kegunaan, integritas, dan keamanan data melalui proses, peran, standar, dan metrik. Komponennya meliputi visi, kebijakan, prosedur, peran, kualitas data, protokol keamanan, audit, monitoring, pelatihan, dan komunikasi.

Mengapa *governance* penting?

Tanpa *governance*, organisasi akan menghadapi masalah berikut:

- data punya banyak versi;
- KPI dihitung dengan cara berbeda;
- tidak jelas siapa pemilik data;
- data sensitif terlalu mudah diakses;
- kualitas data buruk tetapi tidak ada yang bertanggung jawab;
- perubahan data tidak terdokumentasi;
- laporan tidak bisa diaudit;
- dashboard berbeda menghasilkan angka berbeda.

Masalah ini tampak administratif, tetapi dampaknya strategis. Direksi bisa mengambil keputusan investasi, efisiensi, proyek, SDM, atau risiko dari angka yang salah.

A. *Data Owner*

Data owner adalah pihak bisnis yang bertanggung jawab atas arti, aturan, penggunaan, dan kualitas data dalam domain tertentu.

Contoh:

Domain Data	Data Owner
Data pelanggan	Kepala layanan pelanggan / pemasaran
Data penjualan	Kepala penjualan
Data biaya	Kepala keuangan
Data proyek	Kepala PMO / operasi
Data pegawai	Kepala SDM
Data vendor	Kepala pengadaan

Tugas utama *data owner*:

- menetapkan definisi bisnis;
- menyetujui aturan data;
- menentukan prioritas perbaikan kualitas;
- menyetujui akses data sensitif;
- bertanggung jawab atas penggunaan data di domainnya.

Catatan kritis: tim TI biasanya bukan *data owner*. Tim TI mengelola sistem. Pemilik makna data adalah unit bisnis.

B. *Data Steward*

Data steward adalah pelaksana tata kelola data harian. Perannya menjembatani bisnis dan teknis.

Tugas *data steward*:

- menjaga *data dictionary*;
- memantau kualitas data;
- menindaklanjuti data kosong atau salah;
- membantu standarisasi kode;
- memastikan definisi KPI dipakai konsisten;
- mengelola isu data lintas unit;
- mendokumentasikan aturan bisnis.

Jika *data owner* adalah pemilik keputusan, maka *data steward* adalah penjaga disiplin operasional data.

C. *Data Custodian*

Data custodian adalah pihak yang bertanggung jawab secara teknis atas penyimpanan, akses, keamanan, backup, dan performa data.

Biasanya peran ini dijalankan oleh:

- *database administrator*;
- *data engineer*;
- *platform engineer*;
- tim infrastruktur;
- tim keamanan informasi.

Tugas *data custodian*:

- menjaga ketersediaan data;
- mengelola hak akses;
- melakukan backup;
- menjaga keamanan teknis;
- memantau performa;
- memastikan proses *ETL/ELT* berjalan;
- menjaga log dan audit trail.

D. *Data Standard*

Data standard adalah aturan baku tentang format, definisi, struktur, dan nilai data.

Contoh standar:

Elemen Data	Standar
Tanggal	YYYY-MM-DD
Kode cabang	3 digit angka
Kode proyek	format PRJ-YYYY-NNN
Nomor pelanggan	unik dan tidak boleh kosong
Mata uang	IDR sebagai default
Jenis kelamin	kode terkontrol sesuai kebijakan organisasi
Status proyek	planned, active, delayed, closed

Standar data mengurangi tafsir ganda. Tanpa standar, integrasi akan mahal dan kualitas data terus buruk.

E. *Data Policy*

Data policy adalah kebijakan tertulis tentang bagaimana data dibuat, diakses, digunakan, dibagikan, disimpan, dan dihapus.

Contoh kebijakan:

- siapa boleh melihat data sensitif;
- berapa lama data historis disimpan;
- apakah data boleh diekspor;
- bagaimana data dikoreksi;
- siapa menyetujui perubahan definisi KPI;
- bagaimana data personal dilindungi;
- bagaimana audit akses dilakukan.

ISO 8000-150:2022 menekankan pentingnya peran dan tanggung jawab dalam manajemen kualitas data, termasuk bukti dokumentasi implementasi dan kerangka peran untuk mengelola kualitas data. (ISO)

F. *Data Access Control*

Data access control memastikan bahwa data hanya dapat diakses oleh pihak yang berwenang.

Prinsip utama:

- **least privilege**: akses hanya sesuai kebutuhan kerja;
- **role-based access**: akses berdasarkan peran;
- **segregation of duties**: pemisahan tugas kritis;
- **audit trail**: semua akses penting tercatat;
- **masking**: data sensitif disamarkan;
- **approval workflow**: akses sensitif harus disetujui.

Contoh matriks akses:

Peran	Data Penjualan	Data Biaya	Data Pegawai	Ekspor Data	Data Sensitif
Direksi	Ya	Ya	Ringkasan	Terbatas	Masked
Manajer Unit	Ya, unit sendiri	Terbatas	Tidak	Terbatas	Masked
Analis	Ya	Terbatas	Tidak	Ya, terbatas	Masked
Data Engineer	Ya	Ya	Terbatas	Tidak	Terbatas
Auditor	Ya	Ya	Ya	Terbatas	Sesuai mandat

Prinsip kritis: data yang terlalu tertutup tidak berguna; data yang terlalu terbuka berbahaya. *Governance* mencari titik seimbang antara **nilai bisnis dan risiko**.

3.2 Metadata

Metadata adalah **data tentang data**. Dalam *Data Warehouse*, metadata menjelaskan arti, asal, struktur, aturan, transformasi, kualitas, dan penggunaan data.

Referensi yang digunakan menyebut metadata sebagai *road-map* menuju *Data Warehouse*. Metadata membantu sistem pendukung keputusan menemukan isi *warehouse*, menjelaskan sumber, tipe data, relasi, transformasi, kualitas, akses, keamanan, dan penggunaan data.

Tanpa metadata, pengguna akan bertanya:

- kolom ini artinya apa?
- angka ini berasal dari sistem mana?
- kapan data diperbarui?
- apakah nilai ini sudah bersih?
- siapa pemilik datanya?
- bagaimana rumus KPI ini dihitung?
- apakah data ini boleh dipakai untuk laporan resmi?

A. *Business Metadata*

Business metadata menjelaskan arti bisnis dari data.

Contoh:

Elemen	Contoh Business Metadata
pelanggan aktif	pelanggan yang bertransaksi minimal sekali dalam 12 bulan
pendapatan bersih	pendapatan setelah retur dan diskon
proyek terlambat	proyek dengan progres aktual di bawah rencana
vendor aktif	vendor yang memiliki kontrak berjalan
biaya aktual	biaya yang sudah terealisasi dan tercatat resmi

Referensi yang digunakan menjelaskan bahwa *business metadata* mencakup definisi bisnis, aturan bisnis, dan kepemilikan data.

B. *Technical Metadata*

Technical metadata menjelaskan aspek teknis data.

Contoh:

Elemen	Contoh Technical Metadata
nama tabel	fact_sales
nama kolom	transaction_amount
tipe data	numeric
panjang data	18,2
sumber	sistem transaksi
frekuensi update	harian
indeks	customer_id, date_id
partisi	berdasarkan bulan transaksi

Technical metadata penting untuk *data engineer*, administrator, dan pengembang *pipeline*.

C. *Operational Metadata*

Operational metadata menjelaskan informasi operasional proses data.

Contoh:

Elemen	Contoh
waktu mulai ETL	01:00
waktu selesai ETL	01:45
jumlah baris diekstrak	2.000.000
jumlah baris gagal	1.250
status job	sukses / gagal
log error	format tanggal tidak valid
durasi proses	45 menit

Referensi yang digunakan menjelaskan bahwa *operational metadata* mencatat informasi proses seperti *data lineage*, jadwal job, dan histori transformasi.

D. *Data Lineage*

Data lineage menunjukkan perjalanan data dari sumber sampai laporan akhir.

Contoh:

```
sales_transaction.csv
        ↓
stg_sales_raw
        ↓
clean_sales
        ↓
fact_sales
        ↓
dm_sales_monthly
        ↓
dashboard_penjualan_nasional
```

Manfaat *data lineage*:

- menelusuri asal angka;
- mendukung audit;
- membantu analisis dampak perubahan;
- mempercepat pencarian kesalahan;
- meningkatkan kepercayaan pengguna;
- mendukung compliance.

Referensi yang digunakan menjelaskan bahwa *data lineage* mencatat asal, transformasi, dan perpindahan data; pengguna dapat mengikuti asal-usul dan dependensi data untuk audit dan analisis dampak.

E. *Data Dictionary*

Data dictionary adalah kamus data. Isinya menjelaskan tabel, kolom, tipe data, definisi, aturan, dan pemilik data.

Contoh sederhana:

Nama Kolom	Definisi	Tipe Data	Aturan	Pemilik
customer_id	ID unik pelanggan	varchar	wajib, unik	Penjualan
transaction_date	tanggal transaksi	date	format YYYY-MM-DD	Penjualan
amount	nilai transaksi	numeric	tidak boleh negatif	Keuangan
branch_code	kode cabang	varchar	harus terdaftar	Operasi
product_id	ID produk	varchar	wajib	Produk

Referensi metadata repository menjelaskan bahwa *data dictionary* adalah kumpulan metadata yang mendeskripsikan atribut data, definisi, tipe data, dan batasan.

Prinsip kritis: jika organisasi tidak memiliki *data dictionary*, maka definisi data hanya hidup di kepala orang. Itu risiko besar.

3.3 Kualitas Data

Kualitas data menentukan apakah *Data Warehouse* layak dipakai untuk keputusan. Data yang buruk dapat menghasilkan analitik yang rapi tetapi salah.

Referensi yang digunakan menjelaskan bahwa kualitas data mencakup dimensi seperti *accuracy*, *consistency*, *timeliness*, dan *auditability*. Referensi lain menambahkan *validity*, *completeness*, *consistency*, dan *uniformity* sebagai aspek penting dalam pembersihan dan jaminan kualitas data. ISO 8000 juga menyediakan kerangka untuk meningkatkan kualitas data dan menentukan karakteristik data yang relevan untuk kebutuhan tertentu. (ISO)

A. *Completeness*

Completeness berarti data penting tidak kosong dan cukup lengkap untuk analisis.

Contoh masalah:

- pelanggan tidak punya nomor ID;
- transaksi tidak punya tanggal;
- proyek tidak punya lokasi;
- invoice tidak punya nilai;
- data vendor tidak punya NPWP / identitas legal;
- data progres proyek tidak punya periode.

Contoh aturan:

```
customer_id tidak boleh kosong
transaction_date tidak boleh kosong
amount tidak boleh kosong
project_id wajib terisi
```

Risiko jika buruk:

- laporan tidak lengkap;
- model prediksi bias;
- audit sulit;
- KPI tidak akurat.

B. *Accuracy*

Accuracy berarti data mendekati nilai sebenarnya.

Contoh masalah:

- nilai transaksi salah input;
- tanggal proyek salah;
- nama vendor salah eja;
- jumlah stok tidak sesuai fisik;
- nilai pembayaran tidak sesuai invoice.

Contoh pemeriksaan:

```
amount >= 0
tanggal transaksi tidak boleh lebih besar dari tanggal hari ini
nilai invoice harus sama dengan nilai pembayaran yang valid
```

Risiko jika buruk:

- keputusan biaya salah;
- laporan pendapatan salah;
- analisis risiko menyesatkan;
- audit menemukan ketidaksesuaian.

C. *Consistency*

Consistency berarti data seragam antar sistem dan antar periode.

Contoh masalah:

Masalah	Contoh
kode cabang berbeda	JKT01, Jakarta-01, 001
status proyek berbeda	aktif, active, berjalan
format tanggal berbeda	01/12/2026 vs 2026-12-01
satuan berbeda	kg, ton, unit
nama pelanggan berbeda	PT ABC, ABC Ltd, ABC Indonesia

Solusi:

- standarisasi kode;
- referensi master data;
- validasi format;
- tabel mapping;
- aturan transformasi resmi.

D. *Timeliness*

Timeliness berarti data tersedia tepat waktu sesuai kebutuhan bisnis.

Contoh kebutuhan:

Kebutuhan	**Target Waktu**
dashboard operasional	harian
laporan direksi	bulanan
monitoring proyek	mingguan
alert risiko	near-real-time
audit tahunan	sesuai periode audit

Data yang akurat tetapi terlambat bisa kehilangan nilai. Untuk keputusan operasional, terlambat satu hari dapat berarti terlambat bertindak.

E. *Uniqueness*

Uniqueness berarti tidak ada duplikasi yang tidak semestinya.

Contoh masalah:

- satu pelanggan muncul tiga kali;
- satu vendor punya beberapa nama;
- invoice yang sama masuk dua kali;
- produk yang sama punya kode berbeda;
- data pegawai ganda.

Contoh pemeriksaan:

```
customer_id harus unik
invoice_number + vendor_id harus unik
project_code harus unik
```

Risiko jika buruk:

- jumlah pelanggan berlebihan;
- penjualan terhitung ganda;
- biaya terlihat lebih besar;
- vendor tampak lebih banyak dari sebenarnya.

F. *Validity*

Validity berarti data mengikuti format, domain, rentang, dan aturan yang sah.

Contoh aturan:

Field	Aturan Validitas
tanggal	format YYYY-MM-DD
email	harus mengikuti pola email
nilai transaksi	tidak boleh negatif
kode cabang	harus ada di tabel referensi
status proyek	hanya planned, active, delayed, closed
bulan	1 sampai 12

Contoh pemeriksaan:

```
status_project IN ('planned', 'active', 'delayed', 'closed')
amount >= 0
branch_code EXISTS IN dim_branch
transaction_date IS VALID DATE
```

Prinsip kritis: kualitas data tidak bisa hanya diperbaiki di akhir. Kualitas data harus dikontrol sejak sumber, *staging*, transformasi, sampai laporan.

3.4 Checklist Kualitas Data

Checklist kualitas data membantu tim melakukan pemeriksaan cepat sebelum data dipakai untuk laporan resmi, audit, atau dashboard eksekutif.

Checklist dasar

Pertanyaan	Risiko Jika Tidak Dicek
Apakah ada duplikasi?	angka bisa terhitung ganda
Apakah field penting kosong?	laporan tidak lengkap
Apakah format tanggal konsisten?	tren waktu salah
Apakah kode cabang valid?	agregasi wilayah salah
Apakah nilai transaksi masuk akal?	KPI keuangan salah
Apakah ID pelanggan unik?	jumlah pelanggan salah
Apakah data sesuai periode laporan?	laporan tidak tepat waktu
Apakah ada nilai negatif yang tidak wajar?	perhitungan salah
Apakah satuan data konsisten?	agregasi menyesatkan
Apakah data sensitif terlindungi?	risiko hukum dan reputasi

Contoh aturan kualitas data

Dimensi	Aturan	Contoh SQL / Logika
Completeness	tanggal transaksi wajib ada	transaction_date IS NOT NULL
Accuracy	nilai transaksi tidak negatif	amount >= 0
Consistency	kode cabang harus valid	branch_code ada di tabel cabang
Timeliness	data H-1 sudah masuk	max(transaction_date) = kemarin
Uniqueness	invoice tidak boleh ganda	invoice_number unik per vendor
Validity	status harus sesuai daftar	status IN daftar resmi

Hands-on Mini. Membuat Matriks *Governance*, Metadata, dan Kualitas Data

Tujuan praktik

Peserta mampu membuat kontrol awal untuk memastikan data dalam *Data Warehouse* dapat dipercaya.

Langkah 1 — Pilih satu domain data

Contoh: **data transaksi penjualan**.

Langkah 2 — Tentukan peran governance

Komponen	Isi
Domain data	Transaksi penjualan
Data owner	Kepala Penjualan
Data steward	Analis Penjualan
Data custodian	Tim Data Engineering
Pengguna utama	Direksi, manajer penjualan, analis
Risiko utama	transaksi ganda, tanggal salah, nilai kosong

Langkah 3 — Buat metadata sederhana

Kolom	Definisi Bisnis	Tipe Data	Aturan	Pemilik
sales_id	ID unik transaksi	varchar	wajib, unik	Penjualan
customer_id	ID pelanggan	varchar	wajib	Penjualan
branch_code	kode cabang	varchar	wajib, valid	Operasi
transaction_date	tanggal transaksi	date	wajib	Penjualan
amount	nilai transaksi	numeric	>= 0	Keuangan

Langkah 4 — Buat aturan kualitas data

Pemeriksaan	Aturan	Status
Duplikasi transaksi	sales_id harus unik	
Field kosong	sales_id, date, amount wajib terisi	
Format tanggal	YYYY-MM-DD	
Validitas cabang	branch_code ada di master cabang	
Nilai transaksi	amount tidak negatif	
Ketepatan waktu	data kemarin sudah masuk sebelum 07:00	

Langkah 5 — Tentukan tindakan jika data gagal

Masalah	Tindakan
duplikasi	karantina data, kirim ke data steward
field wajib kosong	tolak load ke fact table
kode cabang tidak valid	masuk tabel error, minta koreksi master
nilai transaksi negatif	cek apakah retur atau salah input
data terlambat	kirim notifikasi ke pemilik sistem sumber

Ringkasan Bab

Data Governance, metadata, dan kualitas data adalah fondasi *Data Warehouse* profesional. Tanpa ketiganya, organisasi hanya memindahkan kekacauan data dari sistem sumber ke tempat yang lebih mahal dan lebih cepat diakses.

Poin penting:

- **Data harus punya pemilik, aturan, standar, dan kontrol akses.**
- *Data owner* bertanggung jawab atas makna dan kualitas data.
- *Data steward* menjaga disiplin data harian.
- *Data custodian* menjaga aspek teknis, akses, backup, dan keamanan.
- Metadata menjelaskan arti, struktur, asal, transformasi, dan penggunaan data.
- *Data lineage* penting untuk audit dan analisis dampak.
- Kualitas data harus diuji melalui *completeness*, *accuracy*, *consistency*, *timeliness*, *uniqueness*, dan *validity*.
- Checklist kualitas data harus menjadi bagian dari proses *ETL/ELT*, bukan pekerjaan tambahan di akhir.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Jelaskan perbedaan peran *data owner*, *data steward*, dan *data custodian*.
2. Mengapa metadata disebut sebagai peta jalan menuju *Data Warehouse*?
3. Apa perbedaan *business metadata*, *technical metadata*, dan *operational metadata*?
4. Mengapa *data lineage* penting untuk audit?
5. Jelaskan enam dimensi kualitas data: *completeness*, *accuracy*, *consistency*, *timeliness*, *uniqueness*, dan *validity*.

B. Studi Kasus

Sebuah organisasi memiliki dashboard penjualan nasional. Setelah dicek, jumlah transaksi berbeda antara laporan keuangan dan dashboard penjualan. Beberapa cabang memakai kode berbeda, ada transaksi ganda, dan beberapa nilai transaksi kosong.

Tugas:

- identifikasi masalah governance;
- tentukan siapa *data owner*, *data steward*, dan *data custodian*;
- buat 5 aturan kualitas data;
- buat contoh metadata untuk 5 kolom utama;
- jelaskan risiko bisnis jika masalah tidak diperbaiki.

C. Praktik

Buat tabel berikut untuk satu domain data di organisasi Anda:

Domain Data	Data Owner	Data Steward	Data Custodian	Risiko Utama	Aturan Kualitas Utama

BAGIAN II — DESAIN, MODELING, ETL/ELT, DAN IMPLEMENTASI

BAB 4. Analisis Kebutuhan dan Perencanaan Data Warehouse

Banyak proyek *Data Warehouse* gagal bukan karena teknologi yang dipilih buruk, tetapi karena **kebutuhan bisnis tidak diterjemahkan dengan benar**. Tim teknis membangun tabel, *pipeline*, dan *dashboard*, tetapi pengguna tetap bertanya: “Angka ini menjawab keputusan apa?”

Bab ini membahas tahap paling penting sebelum desain teknis: **analisis kebutuhan dan perencanaan**. Pada tahap ini, organisasi harus menjawab pertanyaan dasar: laporan apa yang diperlukan, siapa penggunanya, keputusan apa yang didukung, data apa yang dibutuhkan, seberapa sering data diperbarui, dan risiko apa yang harus dikendalikan.

Referensi *Mastering Data Warehousing* menempatkan *requirement gathering and analysis* sebagai tahap penting untuk memahami kebutuhan pengguna akhir, proses bisnis yang akan didukung, jenis data yang dibutuhkan, frekuensi pembaruan data, format laporan, dan kualitas sumber data. Pendekatan Kimball juga menekankan pengembangan *Data Warehouse* secara bertahap melalui proses bisnis inti dan *conformed dimensions*, sehingga proyek dapat dipecah menjadi bagian yang lebih mudah dikelola. (Kimball Group)

Golden Rule: *Data Warehouse* yang baik tidak dimulai dari tabel, tetapi dari **keputusan bisnis yang harus dibuat lebih cepat, lebih akurat, dan lebih dapat diaudit**.

4.1 Menentukan Tujuan Bisnis

Tahap pertama adalah menentukan **mengapa *Data Warehouse* dibangun**. Tujuan bisnis harus jelas, spesifik, dan dapat diuji. Tanpa tujuan bisnis, proyek akan melebar, mahal, dan sulit selesai.

Tujuan bisnis harus menjawab lima pertanyaan:

- **Laporan apa yang dibutuhkan?**
- **Keputusan apa yang ingin didukung?**
- **Siapa pengguna utama?**
- **Apa KPI utama?**
- **Seberapa sering data harus diperbarui?**

A. Laporan Apa yang Dibutuhkan?

Laporan adalah bentuk nyata dari kebutuhan informasi. Namun, jangan langsung mengumpulkan semua laporan yang ada. Banyak laporan lama dibuat karena kebiasaan, bukan karena benar-benar dipakai untuk keputusan.

Contoh laporan yang layak masuk prioritas:

Jenis Laporan	Tujuan
Laporan penjualan bulanan	memantau pertumbuhan pendapatan
Laporan biaya proyek	mengendalikan *cost overrun*
Laporan performa cabang	membandingkan kinerja wilayah
Laporan stok kritis	menghindari kekurangan barang
Laporan vendor	mengevaluasi kinerja pemasok
Laporan SLA layanan	memantau kualitas layanan
Laporan audit data	membuktikan asal dan perubahan data

Pertanyaan kritis untuk setiap laporan:

- Apakah laporan ini masih dipakai?
- Siapa yang memakai?
- Keputusan apa yang dibuat dari laporan ini?
- Berapa sering laporan dibutuhkan?
- Apakah definisi angkanya sudah disepakati?
- Apakah laporan ini harus real-time, harian, mingguan, atau bulanan?

Prinsip kritis: laporan yang tidak mempengaruhi keputusan sebaiknya tidak menjadi prioritas awal.

B. Keputusan Apa yang Ingin Didukung?

Data Warehouse harus mendukung keputusan, bukan sekadar menampilkan data. Karena itu, setiap kebutuhan laporan harus dikaitkan dengan keputusan bisnis.

Contoh:

Keputusan Bisnis	Data yang Dibutuhkan
Menutup atau membuka cabang	penjualan, biaya, margin, pelanggan, lokasi
Mengendalikan proyek terlambat	progres, rencana, biaya, vendor, material
Mengurangi biaya operasional	transaksi biaya, unit kerja, kategori biaya
Menentukan produk prioritas	penjualan, margin, stok, tren permintaan
Menilai vendor	ketepatan waktu, kualitas, harga, kontrak
Mendeteksi risiko fraud	transaksi tidak wajar, pola pembayaran, anomali

Dengan cara ini, desain *Data Warehouse* menjadi tajam. Tim tidak sekedar bertanya “data apa yang tersedia?”, tetapi “**keputusan apa yang harus diperbaiki?**”

C. Siapa Pengguna Utama?

Pengguna *Data Warehouse* tidak homogen. Setiap kelompok pengguna memiliki kebutuhan berbeda.

Pengguna	Kebutuhan
Direksi	KPI ringkas, tren, risiko, peringatan dini
Manajer unit	performa unit, target vs realisasi, detail operasional
Analis bisnis	data granular, eksplorasi, filter, segmentasi
Auditor	histori, *data lineage*, kontrol akses, bukti data
Tim TI / Data	kualitas data, proses ETL/ELT, error log
Pengguna dashboard	visualisasi sederhana, cepat, dan konsisten

Kesalahan umum adalah membuat satu *dashboard* untuk semua orang. Padahal direksi membutuhkan ringkasan strategis, sedangkan analis membutuhkan detail dan fleksibilitas.

Prinsip desain: pisahkan kebutuhan **eksekutif**, **manajerial**, **analisis**, dan **audit**.

D. Apa KPI Utama?

KPI adalah jembatan antara strategi dan data. Dalam perencanaan *Data Warehouse*, KPI harus didefinisikan sejak awal.

Contoh KPI:

Domain	KPI
Penjualan	total penjualan, margin, pertumbuhan, pelanggan aktif
Keuangan	realisasi anggaran, biaya per unit, arus kas
Proyek	progres aktual, deviasi jadwal, deviasi biaya
Pengadaan	waktu siklus pengadaan, performa vendor
Layanan	SLA, jumlah keluhan, waktu penyelesaian
SDM	produktivitas, kompetensi, tingkat kehadiran

Setiap KPI harus memiliki:

- nama KPI;
- definisi bisnis;
- rumus;
- sumber data;
- pemilik KPI;
- frekuensi update;
- level detail;
- hak akses;
- aturan validasi.

Contoh definisi KPI:

Elemen	Contoh
Nama KPI	Penjualan Bersih
Definisi	total penjualan setelah diskon dan retur
Rumus	gross sales - discount - return
Sumber	transaksi penjualan, retur, diskon
Pemilik	Divisi Penjualan dan Keuangan
Frekuensi	harian dan bulanan
Risiko	retur terlambat dicatat

Catatan kritis: KPI tanpa definisi resmi akan menciptakan banyak versi kebenaran.

E. Seberapa Sering Data Harus Diperbarui?

Tidak semua data harus diperbarui secara real-time. Banyak organisasi membuang biaya karena semua data dianggap harus cepat. Padahal kebutuhan refresh harus mengikuti keputusan bisnis.

Jenis Kebutuhan	Frekuensi Update
Dashboard direksi	harian / mingguan
Laporan keuangan	harian / bulanan
Monitoring proyek	harian / mingguan
Alert operasional	near-real-time
Audit historis	periodik
Analisis strategis	bulanan / kuartalan

IBM menjelaskan bahwa *ETL* menggabungkan, membersihkan, dan mengorganisasi data dari berbagai sumber ke dalam repositori terpadu seperti *data warehouse*, sedangkan *ELT* dan *ETL* sama-sama memindahkan data mentah dari sumber ke target tetapi berbeda pada urutan transformasinya. (IBM)

Prinsip kritis: semakin sering data diperbarui, semakin tinggi biaya, kompleksitas, dan risiko operasional. Jangan memilih real-time jika keputusan bisnis hanya bulanan.

4.2 Identifikasi *Stakeholder*

Stakeholder adalah pihak yang mempengaruhi, menggunakan, menyetujui, mengelola, atau diaudit berdasarkan data. Identifikasi *stakeholder* penting karena *Data Warehouse* adalah proyek lintas fungsi.

Referensi *Mastering Data Warehousing* menekankan pentingnya menyusun *project charter*, membentuk tim, serta mengidentifikasi *stakeholder* dan pengambil keputusan sejak awal proyek.

A. Direksi

Direksi membutuhkan data untuk keputusan strategis.

Kebutuhan direksi:

- KPI utama;
- tren bisnis;
- risiko utama;
- performa unit;
- target vs realisasi;
- peringatan dini;
- ringkasan keuangan;
- insight strategis.

Bahasa direksi bukan bahasa tabel. Bahasa direksi adalah **risiko, dampak, prioritas, biaya, waktu, dan keputusan**.

B. Manajer Unit

Manajer unit membutuhkan data untuk mengendalikan operasi.

Kebutuhan manajer:

- performa unit;
- pencapaian target;
- penyebab deviasi;
- daftar masalah;
- detail transaksi;
- tindak lanjut operasional;
- perbandingan antar periode.

Manajer biasanya membutuhkan kombinasi ringkasan dan detail. Mereka harus bisa turun dari KPI ke penyebab.

C. Analis Bisnis

Analis bisnis membutuhkan fleksibilitas untuk eksplorasi.

Kebutuhan analis:

- data granular;
- dimensi analisis;
- filter;
- data historis;
- akses query;
- dokumentasi definisi;
- metadata;
- kemampuan ekspor terbatas.

Analis dapat membantu menemukan pola, anomali, dan peluang. Namun akses analis harus tetap dikontrol agar tidak menimbulkan risiko data sensitif.

D. Tim TI / Tim Data

Tim teknis bertanggung jawab menterjemahkan kebutuhan bisnis menjadi desain data, proses, keamanan, dan operasi.

Kebutuhan tim teknis:

- daftar sumber data;
- struktur data;
- volume data;
- format data;
- frekuensi update;
- aturan transformasi;
- kebutuhan keamanan;
- SLA;
- retensi;
- kebutuhan backup;
- kebutuhan monitoring.

E. Auditor

Auditor membutuhkan bukti, bukan hanya angka.

Kebutuhan auditor:

- asal data;
- *data lineage*;
- log perubahan;
- kontrol akses;
- bukti rekonsiliasi;
- histori data;
- definisi KPI;
- dokumentasi proses.

Untuk organisasi yang serius, kebutuhan audit harus dirancang sejak awal, bukan ditambal setelah sistem berjalan.

F. *Data Owner*

Data owner adalah pemilik makna dan aturan data. Mereka harus menyetujui definisi data dan KPI.

Contoh:

Domain	***Data Owner***
Penjualan	Kepala Penjualan
Biaya	Kepala Keuangan
Proyek	Kepala Operasi / PMO
Vendor	Kepala Pengadaan
Pegawai	Kepala SDM
Layanan	Kepala Layanan

G. Pengguna *Dashboard*

Pengguna *dashboard* adalah pihak yang akan memakai hasil akhir sistem.

Yang harus ditanyakan:

- dashboard dipakai untuk rapat apa?
- seberapa sering dibuka?
- KPI apa yang harus tampil paling atas?
- filter apa yang dibutuhkan?
- apakah perlu akses detail?
- apakah perlu notifikasi?
- apakah data boleh diekspor?

Prinsip kritis: pengguna *dashboard* sering tidak bisa menjelaskan kebutuhan teknis, tetapi mereka tahu keputusan apa yang sulit dibuat hari ini.

4.3 Analisis Sumber Data

Setelah kebutuhan bisnis jelas, tahap berikutnya adalah menganalisis sumber data. Tujuannya adalah mengetahui apakah data yang dibutuhkan **tersedia, cukup berkualitas, dapat diakses, dan legal digunakan**.

Referensi *Data Mining and Warehousing Techniques with Machine Learning Concepts* menekankan bahwa desain *Data Warehouse* perlu mengidentifikasi sumber data internal dan eksternal, memahami struktur serta format sumber data, menilai kualitas data, serta menyiapkan tata kelola dan keamanan.

A. Sistem Transaksi

Sistem transaksi adalah sumber utama data operasional. Contoh:

- transaksi penjualan;
- invoice;
- pembayaran;
- pengadaan;
- stok;
- proyek;
- absensi;
- layanan pelanggan.

Hal yang perlu dianalisis:

- nama sistem;
- pemilik sistem;
- struktur tabel;
- volume data;
- frekuensi perubahan;
- ketersediaan histori;
- kualitas data;
- metode akses;
- dampak ke sistem operasional jika di ekstrak.

Catatan kritis: jangan sampai proses ekstraksi data membebani sistem transaksi utama.

B. File CSV / Spreadsheet

Banyak organisasi masih menyimpan data penting dalam file. Data seperti ini harus diperlakukan hati-hati.

Risiko umum:

- format berubah-ubah;
- nama kolom tidak konsisten;
- data manual rawan salah;
- tidak ada validasi;
- file ganda;
- versi tidak jelas;
- pemilik data tidak jelas.

Namun, file tetap dapat dipakai sebagai sumber awal jika ada aturan:

- template baku;
- lokasi penyimpanan resmi;
- format kolom tetap;
- validasi otomatis;
- tanggal versi;
- pemilik data;
- proses approval.

C. API

API berguna untuk menarik data dari aplikasi modern atau layanan eksternal.

Yang harus dinilai:

- endpoint yang tersedia;
- batas akses;
- autentikasi;
- format data;
- stabilitas API;
- dokumentasi;
- batas frekuensi akses;
- histori data;
- biaya penggunaan;
- risiko perubahan skema.

API cocok untuk integrasi yang terjadwal dan terkontrol, tetapi tetap perlu *logging* dan validasi.

D. Log Aplikasi

Log aplikasi dapat menjadi sumber analitik penting.

Contoh penggunaan:

- analisis perilaku pengguna;
- audit akses;
- deteksi error;
- monitoring performa;
- analisis fraud;
- analisis layanan digital.

Risiko log:

- volume besar;
- format tidak stabil;
- data tidak terstruktur;
- banyak noise;
- data sensitif bisa ikut tercatat;
- retensi pendek.

Log sebaiknya dipakai untuk analitik tertentu, bukan langsung dicampur dengan data transaksi tanpa desain yang jelas.

E. Data Eksternal

Data eksternal dapat memperkaya analisis.

Contoh:

- data ekonomi;
- data cuaca;
- data wilayah;
- data demografi;
- data harga komoditas;
- data lalu lintas;
- data pasar;
- data regulasi.

Yang harus dicek:

- sumber resmi;
- lisensi penggunaan;
- frekuensi update;
- format;
- reliabilitas;
- biaya;
- metode integrasi;
- relevansi bisnis.

F. Data Historis

Data historis adalah kekuatan utama *Data Warehouse*. Namun data historis sering memiliki masalah:

- format lama berbeda;
- kode lama tidak cocok dengan kode baru;
- data tidak lengkap;
- ada sistem lama yang sudah tidak aktif;
- histori hanya tersedia dalam file;
- dokumentasi hilang.

Langkah penting:

- tentukan periode histori yang dibutuhkan;
- identifikasi format lama;
- buat mapping kode lama ke kode baru;
- dokumentasikan asumsi;
- pisahkan data yang tidak bisa divalidasi;
- lakukan rekonsiliasi.

Template Analisis Sumber Data

Sumber Data	Pemilik	Format	Volume	Frekuensi Update	Kualitas	Metode Akses	Risiko
Transaksi penjualan	Penjualan	Database	2 juta/bulan	Harian	Sedang	Replikasi / ekspor	beban sistem
Invoice	Keuangan	Database	200 ribu/bulan	Harian	Tinggi	Query terjadwal	akses sensitif
Target penjualan	Manajemen	Spreadsheet	kecil	Bulanan	Sedang	Upload file	format berubah
Data pelanggan	Layanan	Database	1 juta	Harian	Rendah-Sedang	API	duplikasi
Log aplikasi	TI	File log	besar	Real-time	Sedang	Stream / batch	volume besar

4.4 Menentukan *Scope* Awal

Kesalahan terbesar proyek *Data Warehouse* adalah ingin membangun semuanya sekaligus. Akibatnya, proyek menjadi terlalu luas, lambat, dan sulit dikendalikan.

Referensi yang digunakan menyebut *incremental development* sebagai pendekatan yang membangun *Data Warehouse* secara bertahap, memungkinkan perbaikan berkelanjutan dan umpan balik dari pengguna. Kimball Group juga menekankan arsitektur yang memungkinkan pengembangan *Data Warehouse* dan *BI* secara inkremental dengan memecah proses perencanaan ke bagian yang lebih dapat dikelola. (Kimball Group)

A. Mulai dari 1–2 Domain Data

Domain data adalah area bisnis yang relatif jelas.

Contoh domain:

- penjualan;
- keuangan;
- proyek;
- pengadaan;
- layanan pelanggan;
- SDM;
- stok;
- risiko.

Pilih domain awal berdasarkan:

- nilai bisnis tinggi;
- data tersedia;
- sponsor kuat;
- pengguna jelas;
- risiko terkendali;
- quick win terlihat;
- kompleksitas tidak terlalu tinggi.

Contoh prioritas awal:

Domain	Nilai Bisnis	Ketersediaan Data	Kompleksitas	Prioritas
Penjualan	5	5	3	Tinggi
Keuangan	5	4	4	Tinggi
Proyek	5	3	5	Sedang
SDM	3	4	3	Sedang
Pengadaan	4	3	4	Sedang

B. Hindari Membangun Semuanya Sekaligus

Membangun semua domain sejak awal berisiko karena:

- kebutuhan terlalu banyak;
- sumber data terlalu kompleks;
- definisi KPI belum matang;
- tim kewalahan;
- biaya naik;
- *timeline* melebar;
- pengguna lama menunggu hasil;
- risiko gagal meningkat.

Pendekatan yang lebih sehat:

- pilih domain prioritas;
- bangun model awal;
- validasi dengan pengguna;
- rilis dashboard pertama;
- kumpulkan umpan balik;
- perluas domain berikutnya.

C. Gunakan Pendekatan MVP → Pilot → Produksi

Pendekatan bertahap:

```
MVP → Pilot → Produksi → Perluasan
```

1. *Minimum Viable Product* / MVP

Tujuan MVP adalah membuktikan bahwa *Data Warehouse* dapat memberi nilai cepat.

Ciri MVP:

- 1 domain data;
- 3–5 KPI utama;
- 1–2 sumber data;
- data historis terbatas;
- dashboard sederhana;
- validasi manual masih diperbolehkan;
- fokus pada pembuktian nilai.

Contoh MVP:

- dashboard penjualan bulanan;
- dashboard progres proyek utama;
- dashboard biaya per unit;
- dashboard SLA layanan.

2. Pilot

Pilot memperluas MVP ke pengguna nyata.

Ciri pilot:

- pengguna terbatas;
- data lebih lengkap;
- aturan kualitas data mulai formal;
- *data owner* aktif;
- feedback dikumpulkan;
- proses refresh dijalankan rutin;
- error dan gap dicatat.

3. Produksi

Produksi berarti sistem siap dipakai sebagai rujukan resmi.

Syarat produksi:

- definisi KPI disetujui;
- kualitas data diuji;
- akses pengguna diatur;
- metadata tersedia;
- proses refresh stabil;
- backup aktif;
- dokumentasi lengkap;
- UAT selesai;
- pemilik operasional jelas.

D. Kriteria *Scope* yang Baik

Scope awal harus cukup kecil untuk selesai, tetapi cukup penting untuk bernilai.

Ciri *scope* yang baik:

- masalah bisnis jelas;
- pengguna jelas;
- KPI jelas;
- sumber data tersedia;
- pemilik data ada;
- risiko dapat dikendalikan;
- hasil dapat dilihat dalam waktu singkat;
- dapat diperluas ke fase berikutnya.

Prinsip kritis: *scope* kecil yang selesai lebih berharga daripada *scope* besar yang tidak pernah produksi.

4.5 Dokumen Kebutuhan

Dokumen kebutuhan adalah jembatan antara bisnis dan teknis. Dokumen ini tidak perlu panjang, tetapi harus jelas, terstruktur, dan dapat ditandatangani oleh pihak terkait.

Output utama:

- *Business requirement*
- *Data requirement*
- *Security requirement*
- *Service Level Agreement* / SLA
- *Data refresh requirement*
- *Retention requirement*

A. *Business Requirement*

Business requirement menjelaskan kebutuhan bisnis yang harus didukung.

Isi minimal:

- tujuan bisnis;
- masalah yang ingin diselesaikan;
- pengguna utama;
- laporan utama;
- KPI utama;
- keputusan yang didukung;
- prioritas;
- batasan awal;
- kriteria sukses.

Contoh:

Elemen	Isi
Tujuan	meningkatkan visibilitas penjualan nasional
Masalah	laporan cabang berbeda format dan lambat
Pengguna	direksi, kepala penjualan, analis
KPI	penjualan, margin, pertumbuhan, pelanggan aktif
Keputusan	prioritas cabang, promosi, target wilayah
Kriteria sukses	dashboard bulanan disetujui pengguna

B. *Data Requirement*

Data requirement menjelaskan data yang dibutuhkan untuk memenuhi kebutuhan bisnis.

Isi minimal:

- sumber data;
- tabel / file;
- kolom penting;
- periode data;
- volume;
- frekuensi update;
- aturan transformasi;
- kualitas data;
- data historis;
- pemilik data.

Contoh:

Kebutuhan	Detail
Sumber	transaksi penjualan, pelanggan, produk, cabang
Periode historis	3 tahun
Frekuensi update	harian
Kolom utama	tanggal, produk, pelanggan, cabang, nilai
Transformasi	standarisasi kode cabang
Kualitas	deteksi duplikasi dan nilai kosong

C. *Security Requirement*

Security requirement menjelaskan bagaimana data dilindungi.

Isi minimal:

- klasifikasi data;
- hak akses;
- data sensitif;
- kebutuhan masking;
- audit log;
- aturan ekspor;
- persetujuan akses;
- enkripsi;
- pembatasan pengguna.

Contoh:

Area	Aturan
Data pelanggan	akses terbatas
Data nilai transaksi	hanya role tertentu
Export	hanya analis tertentu
Audit log	semua akses data sensitif dicatat
Masking	nomor identitas / kontak disamarkan

D. SLA

SLA menjelaskan standar layanan teknis dan operasional.

Contoh SLA:

Elemen SLA	Target
Ketersediaan dashboard	99% pada jam kerja
Refresh data harian	selesai sebelum 07:00
Waktu respon dashboard	< 10 detik untuk halaman utama
Penanganan error ETL	maksimal 4 jam
Restore data	sesuai target RTO
Kehilangan data maksimum	sesuai target RPO

SLA harus realistis. SLA terlalu tinggi akan menaikkan biaya.

E. *Data Refresh Requirement*

Data refresh requirement menjelaskan kapan dan bagaimana data diperbarui.

Contoh:

Data	Frekuensi	Waktu Proses	Metode
Transaksi penjualan	harian	01:00–03:00	incremental load
Data pelanggan	harian	02:00	incremental load
Target penjualan	bulanan	awal bulan	file upload
Data produk	mingguan	Minggu malam	full refresh
Data cabang	bulanan	awal bulan	manual validated upload

Pertanyaan kritis:

- apakah data harus harian?
- apakah ada kebutuhan near-real-time?
- apakah sistem sumber siap?
- apakah ada jendela proses?
- apa yang terjadi jika refresh gagal?

F. *Retention Requirement*

Retention requirement menjelaskan berapa lama data disimpan.

Contoh:

Jenis Data	Retensi
Transaksi penjualan	5 tahun
Log ETL	1 tahun
Data audit	sesuai kebijakan audit
Data staging mentah	3–6 bulan
Backup	sesuai kebijakan organisasi
Data agregat	5–10 tahun

Retensi harus mempertimbangkan:

- kebutuhan analitik;
- audit;
- regulasi;
- biaya storage;
- risiko data sensitif;
- kebutuhan historis;
- kebijakan penghapusan.

Hands-on. Template Data Warehouse Requirement Brief

Gunakan template ini untuk menyusun kebutuhan awal proyek *Data Warehouse*. Template ini ringkas, tetapi cukup untuk menghubungkan kebutuhan bisnis, data, keamanan, SLA, refresh, dan retensi.

1. Identitas Proyek

Elemen	Isi
Nama proyek	
Sponsor bisnis	
Unit pemilik	
Manajer proyek	
Tanggal dokumen	
Versi dokumen	

2. Ringkasan Kebutuhan Bisnis

Pertanyaan	Jawaban
Masalah bisnis utama	
Keputusan yang ingin didukung	
Pengguna utama	
Domain data awal	
Target implementasi	MVP / Pilot / Produksi
Kriteria sukses	

Contoh isi:

Pertanyaan	Jawaban
Masalah bisnis utama	laporan penjualan nasional lambat dan tidak konsisten
Keputusan yang ingin didukung	penentuan target cabang dan evaluasi produk
Pengguna utama	direksi, kepala penjualan, analis
Domain data awal	penjualan dan pelanggan
Target implementasi	MVP
Kriteria sukses	dashboard penjualan harian tervalidasi oleh bisnis

3. Daftar Laporan dan *Dashboard*

Nama Laporan / Dashboard	Pengguna	Frekuensi	Keputusan yang Didukung	Prioritas

Contoh:

Nama Laporan / Dashboard	Pengguna	Frekuensi	Keputusan yang Didukung	Prioritas
Dashboard Penjualan Nasional	Direksi	Harian	evaluasi target dan tren	Tinggi
Laporan Margin Produk	Penjualan / Keuangan	Mingguan	prioritas produk	Tinggi
Laporan Cabang	Manajer wilayah	Bulanan	evaluasi cabang	Sedang

4. KPI Utama

KPI	Definisi	Rumus	Sumber Data	Pemilik	Frekuensi

Contoh:

KPI	Definisi	Rumus	Sumber Data	Pemilik	Frekuensi
Penjualan Bersih	penjualan setelah diskon dan retur	gross - diskon - retur	transaksi, retur	Penjualan / Keuangan	Harian
Margin	selisih penjualan dan HPP	penjualan - HPP	transaksi, produk	Keuangan	Mingguan
Pelanggan Aktif	pelanggan yang bertransaksi 12 bulan terakhir	count distinct customer_id	transaksi	Penjualan	Bulanan

5. Analisis Sumber Data

Sumber Data	Pemilik	Format	Volume	Frekuensi Update	Metode Akses	Risiko Kualitas

Contoh:

Sumber Data	Pemilik	Format	Volume	Frekuensi Update	Metode Akses	Risiko Kualitas
Transaksi penjualan	Penjualan	Database	2 juta/bulan	Harian	query terjadwal	Sedang
Data pelanggan	Layanan	Database	1 juta	Harian	API	Tinggi
Target penjualan	Manajemen	Spreadsheet	kecil	Bulanan	upload file	Sedang

6. Kebutuhan Keamanan

Jenis Data	Klasifikasi	Pengguna yang Boleh Akses	Kontrol

Contoh:

Jenis Data	Klasifikasi	Pengguna yang Boleh Akses	Kontrol
Data pelanggan	sensitif	analis terbatas	masking
Nilai transaksi	internal	manajemen dan analis	role-based access
Data cabang	internal	manajer wilayah	filter berdasarkan wilayah
Log akses	audit	auditor dan admin	read-only

7. Kebutuhan Refresh Data

Data	Frekuensi	Waktu Proses	Jenis Load	Resiko Jika Gagal

Contoh:

Data	Frekuensi	Waktu Proses	Jenis Load	Resiko Jika Gagal
Transaksi penjualan	harian	01:00–03:00	incremental	dashboard tidak update
Produk	mingguan	Minggu malam	full refresh	kategori produk salah
Target penjualan	bulanan	awal bulan	upload file	KPI target tidak valid

8. SLA

Elemen	Target
Ketersediaan dashboard	
Waktu refresh	
Waktu respon query	
Penanganan error	
Backup	
Restore	

Contoh:

Elemen	Target
Ketersediaan dashboard	99% pada jam kerja
Waktu refresh	selesai sebelum 07:00
Waktu respon query	< 10 detik untuk halaman utama
Penanganan error	maksimal 4 jam
Backup	harian
Restore	diuji berkala

9. Retensi Data

Jenis Data	Retensi	Alasan

Contoh:

Jenis Data	Retensi	Alasan
Transaksi penjualan	5 tahun	analitik historis dan audit
Staging mentah	3 bulan	troubleshooting
Log ETL	1 tahun	audit proses
Data agregat	5 tahun	tren strategis

10. Batasan *Scope* Awal

Area	Masuk Scope	Tidak Masuk Scope
Domain data		
Laporan		
Pengguna		
Periode data		
Integrasi		

Contoh:

Area	Masuk Scope	Tidak Masuk Scope
Domain data	penjualan, pelanggan	pengadaan, SDM
Laporan	dashboard penjualan	prediksi AI
Pengguna	direksi, penjualan	seluruh cabang
Periode data	3 tahun	lebih dari 5 tahun
Integrasi	transaksi dan pelanggan	data eksternal

11. Risiko Awal

Risiko	Dampak	Mitigasi
data pelanggan duplikat	jumlah pelanggan salah	deduplikasi dan master data
format spreadsheet berubah	ETL gagal	template baku
data terlambat	dashboard tidak akurat	alert dan SLA sumber data
definisi KPI berbeda	angka diperdebatkan	persetujuan *data owner*
akses sensitif terlalu luas	risiko keamanan	kontrol akses dan masking

12. Persetujuan

Peran	Nama	Status
Sponsor bisnis		
Data owner		
Perwakilan pengguna		
Tim data		
Auditor / compliance		

Checklist Bab 4 untuk Sertifikasi

Pertanyaan	Ya/Tidak
Apakah tujuan bisnis sudah jelas?	
Apakah keputusan yang didukung sudah disebutkan?	
Apakah pengguna utama sudah dipetakan?	
Apakah KPI sudah memiliki definisi dan rumus?	
Apakah sumber data sudah dianalisis?	
Apakah kualitas data awal sudah dinilai?	
Apakah kebutuhan keamanan sudah ditulis?	
Apakah SLA sudah realistis?	
Apakah kebutuhan refresh data sudah jelas?	
Apakah retensi data sudah ditentukan?	
Apakah scope awal tidak terlalu luas?	
Apakah ada rencana MVP → pilot → produksi?	

Kesalahan Umum dalam Analisis Kebutuhan

Kesalahan yang sering terjadi:

- mulai dari teknologi, bukan keputusan;
- semua laporan dianggap penting;
- KPI tidak didefinisikan;
- *data owner* tidak dilibatkan;
- kualitas data sumber tidak dicek;
- *scope* terlalu luas;
- kebutuhan audit terlambat dipikirkan;
- SLA dibuat terlalu tinggi;
- kebutuhan refresh dibuat terlalu sering;
- retensi data tidak dihitung;
- pengguna akhir tidak ikut UAT.

Prinsip kritis: kebutuhan yang kabur akan menghasilkan arsitektur yang kabur, biaya yang kabur, dan hasil yang sulit dipercaya.

Ringkasan Bab

Analisis kebutuhan adalah tahap yang menentukan arah *Data Warehouse*. Tujuannya adalah menerjemahkan kebutuhan bisnis menjadi kebutuhan teknis yang jelas, terukur, dan dapat dibangun.

Poin penting:

- **Mulai dari keputusan bisnis, bukan dari tabel.**
- Laporan harus dikaitkan dengan keputusan yang didukung.
- KPI harus punya definisi, rumus, sumber, pemilik, dan frekuensi.
- *Stakeholder* harus dipetakan sejak awal.
- Sumber data harus dianalisis dari sisi format, volume, kualitas, akses, dan risiko.
- *Scope* awal harus kecil, tajam, dan bernilai.
- Pendekatan MVP → pilot → produksi mengurangi risiko.
- Dokumen kebutuhan harus mencakup *business requirement*, *data requirement*, keamanan, SLA, refresh, dan retensi.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Mengapa analisis kebutuhan harus dimulai dari keputusan bisnis?
2. Apa perbedaan *business requirement* dan *data requirement*?
3. Mengapa KPI harus memiliki definisi resmi?
4. Apa risiko jika *scope* awal terlalu luas?
5. Jelaskan perbedaan MVP, pilot, dan produksi dalam proyek *Data Warehouse*.

B. Studi Kasus

Sebuah perusahaan ingin membuat *Data Warehouse* untuk memantau penjualan nasional. Data berasal dari transaksi cabang, data pelanggan, data produk, target penjualan, dan invoice. Direksi ingin melihat penjualan harian, margin produk, performa cabang, dan pelanggan aktif.

Tugas:

- tentukan 5 KPI utama;
- identifikasi minimal 5 sumber data;
- tentukan pengguna utama;
- buat *scope* MVP;
- tentukan frekuensi refresh data;
- buat 3 risiko awal dan mitigasinya.

C. Praktik

Buat *Data Warehouse Requirement Brief* untuk satu domain data di organisasi Anda dengan struktur:

Bagian	Isi
Tujuan bisnis	
Keputusan yang didukung	
Pengguna utama	
KPI utama	
Sumber data	
Security requirement	
SLA	
Refresh requirement	
Retention requirement	
Scope MVP	

BAB 5. Dimensional Modeling: Fact, Dimension, Star Schema, dan Snowflake Schema

Dimensional modeling adalah inti desain *Data Warehouse*. Jika Bab 4 menjawab “kebutuhan bisnis apa yang harus didukung?”, maka Bab 5 menjawab “bagaimana kebutuhan itu diterjemahkan menjadi struktur data analitik yang rapi?”

Dalam sistem operasional, data biasanya dirancang agar transaksi cepat dan konsisten. Dalam *Data Warehouse*, desain diarahkan agar pengguna dapat menjawab pertanyaan bisnis dengan cepat:

- Berapa penjualan per bulan?
- Produk mana yang paling menguntungkan?
- Cabang mana yang menurun?
- Pelanggan mana yang paling bernilai?
- Proyek mana yang biayanya melewati rencana?
- Wilayah mana yang perlu perhatian manajemen?

Referensi yang digunakan menegaskan bahwa *dimensional modeling* mengorganisasi data ke dalam *fact table* dan *dimension table*. *Fact table* menyimpan ukuran kuantitatif, sedangkan *dimension table* memberi konteks seperti waktu, produk, pelanggan, atau lokasi. Model ini banyak digunakan karena cocok untuk *query*, analisis, dan pelaporan. Kimball Group juga menempatkan *dimensional modeling* sebagai kumpulan teknik inti untuk *fact table*, *dimension table*, *conformed dimension*, dan *slowly changing dimension*. (Kimball Group)

Golden Rule: *Dimensional modeling* yang baik selalu dimulai dari **proses bisnis dan grain**, bukan dari nama tabel.

5.1 Konsep *Dimensional Modeling*

Dimensional modeling adalah teknik desain data untuk analitik. Tujuannya adalah membuat struktur data yang:

- mudah dipahami pengguna bisnis;
- cepat untuk *query*;
- cocok untuk agregasi;
- mendukung *dashboard*;
- mampu menyimpan histori;
- mudah diperluas;
- jelas hubungan antar tabelnya.

Dalam model dimensional, ada dua kelompok tabel utama:

- ***Fact table***: menyimpan kejadian bisnis dan angka yang diukur.
- ***Dimension table***: menyimpan konteks untuk memahami angka tersebut.

Contoh sederhana:

```
FactSales
  ↓
DimCustomer
DimProduct
DimDate
DimRegion
DimSalesChannel
```

Tabel *fact* menjawab “berapa?”, sedangkan tabel *dimension* menjawab “berdasarkan apa?”

A. *Fact Table*

Fact table adalah tabel pusat yang menyimpan kejadian bisnis yang dapat dihitung atau dianalisis. Biasanya berisi angka dan kunci yang mengarah ke tabel dimensi.

Contoh *fact table*:

- *FactSales*
- *FactInvoice*
- *FactProjectProgress*
- *FactProjectCost*
- *FactEnergyUsage*
- *FactQHSEIncident*

Contoh kolom dalam *FactSales*:

Kolom	Fungsi
sales_id	identitas transaksi
date_key	hubungan ke DimDate
customer_key	hubungan ke DimCustomer
product_key	hubungan ke DimProduct
region_key	hubungan ke DimRegion
sales_channel_key	hubungan ke DimSalesChannel
quantity	jumlah barang
gross_amount	nilai penjualan kotor
discount_amount	diskon
net_amount	penjualan bersih
cost_amount	biaya pokok
margin_amount	margin

Fact table biasanya memiliki baris sangat banyak karena menyimpan kejadian bisnis. Referensi yang digunakan menyebut *fact table* sebagai tabel inti dalam model dimensional yang berisi ukuran kuantitatif seperti penjualan, margin, atau jumlah unit, serta kunci ke tabel dimensi.

Prinsip kritis: jangan memasukkan terlalu banyak deskripsi ke *fact table*. Deskripsi harus masuk ke *dimension table*. *Fact table* harus fokus pada kejadian, ukuran, dan kunci.

B. *Dimension Table*

Dimension table memberi konteks terhadap angka di *fact table*. Jika *fact table* menyimpan “nilai penjualan”, maka *dimension table* menjelaskan penjualan itu terjadi kapan, oleh siapa, untuk produk apa, di wilayah mana, dan melalui kanal apa.

Contoh *dimension table*:

- *DimCustomer*
- *DimProduct*
- *DimDate*
- *DimRegion*
- *DimSalesChannel*
- *DimProject*
- *DimVendor*
- *DimEmployee*

Contoh kolom dalam *DimProduct*:

Kolom	Fungsi
product_key	surrogate key
product_id	kode produk dari sistem sumber
product_name	nama produk
category	kategori
subcategory	subkategori
brand	merek
unit	satuan
status	aktif / tidak aktif

Referensi menyebut *dimension table* sebagai tabel yang menyimpan informasi deskriptif, tekstual, atau kategorikal yang memberi konteks pada ukuran di *fact table*.

Prinsip kritis: dimensi harus dibuat dengan bahasa bisnis. Pengguna harus bisa memahami isi dimensi tanpa membaca struktur sistem operasional.

C. *Measure*

Measure adalah angka yang diukur dalam *fact table*. Biasanya digunakan untuk agregasi, perbandingan, tren, dan KPI.

Contoh *measure*:

Domain	**Contoh Measure**
Penjualan	quantity, gross_amount, net_amount, margin
Keuangan	invoice_amount, payment_amount, outstanding_amount
Proyek	planned_progress, actual_progress, cost_variance
Energi	kWh, peak_load, cost_energy
Layanan	response_time, resolution_time, complaint_count

Tidak semua angka otomatis menjadi *measure*. Beberapa angka bisa menjadi atribut dimensi.

Contoh:

- umur pelanggan bisa menjadi atribut di *DimCustomer*;
- jumlah transaksi adalah *measure*;
- kode pos adalah atribut dimensi;
- nilai penjualan adalah *measure*.

Aturan praktis: jika angka dijumlahkan, dirata-ratakan, dihitung trennya, atau dibandingkan sebagai KPI, kemungkinan besar itu *measure*.

D. *Grain*

Grain adalah tingkat detail satu baris dalam *fact table*. Ini keputusan desain paling penting dalam *dimensional modeling*.

Contoh *grain*:

- 1 baris = 1 transaksi penjualan.
- 1 baris = 1 item transaksi penjualan.
- 1 baris = 1 invoice.
- 1 baris = 1 pembayaran.
- 1 baris = 1 progres proyek per minggu.
- 1 baris = 1 pemakaian energi per jam.

Referensi yang digunakan menjelaskan bahwa *grain* menentukan tingkat detail fakta dalam *fact table*, misalnya penjualan harian per produk per toko.

Kesalahan fatal: membuat *fact table* tanpa mendefinisikan *grain*. Jika *grain* tidak jelas, angka bisa terhitung ganda, agregasi salah, dan dashboard tidak dapat dipercaya.

E. *Hierarchy*

Hierarchy adalah susunan level analisis dalam dimensi. Hierarki memungkinkan pengguna melakukan *roll-up* dan *drill-down*.

Contoh hirarki waktu:

```
Tanggal → Bulan → Kuartal → Tahun
```

Contoh hirarki wilayah:

```
Cabang → Kota → Provinsi → Region → Nasional
```

Contoh hirarki produk:

```
Produk → Sub Kategori → Kategori → Divisi Produk
```

Contoh hirarki proyek:

```
Aktivitas → Paket Pekerjaan → Proyek → Program
```

Hierarki penting untuk *OLAP* karena pengguna sering mulai dari ringkasan, lalu turun ke detail.

F. *Surrogate Key*

Surrogate key adalah kunci buatan di *Data Warehouse*. Ia berbeda dari *natural key* yang berasal dari sistem operasional.

Contoh:

Jenis Key	Contoh
Natural key	customer_id = CUST-001
Surrogate key	customer_key = 10001

Mengapa *surrogate key* penting?

- menjaga stabilitas saat kode sumber berubah;
- mendukung *Slowly Changing Dimension*;
- menghindari konflik kode antar sistem;
- memudahkan integrasi data;
- menjaga histori versi dimensi.

Referensi menyebut *surrogate key* sebagai identitas unik buatan yang sering digunakan di *Data Warehouse* untuk menjaga integritas data dan mengelola perubahan dari sistem sumber.

PostgreSQL documentation menjelaskan bahwa *foreign key* menjaga integritas referensial dengan memastikan nilai pada suatu kolom harus sesuai dengan nilai yang ada pada tabel referensi. Ini penting untuk relasi antara *fact table* dan *dimension table*. (PostgreSQL)

5.2 Menentukan *Grain*

Menentukan *grain* adalah langkah pertama sebelum membuat tabel. *Grain* menjawab: **satu baris dalam tabel fakta mewakili apa?**

Jika *grain* salah, seluruh desain akan bermasalah. Tidak ada indeks, dashboard, atau visualisasi yang dapat menyelamatkan model yang *grain*-nya kacau.

A. Contoh *Grain* Penjualan

Pilihan 1:

```
1 baris = 1 transaksi penjualan
```

Cocok jika analisis cukup pada level transaksi.

Contoh:

sales_id	date_key	customer_key	net_amount
S001	20260101	101	500000
S002	20260101	102	300000

Pilihan 2:

```
1 baris = 1 item dalam transaksi penjualan
```

Cocok jika organisasi perlu analisis produk per transaksi.

Contoh:

sales_id	line_no	product_key	quantity	net_amount
S001	1	201	2	200000
S001	2	202	1	300000

Pilihan kedua lebih detail dan lebih fleksibel, tetapi jumlah data lebih besar.

B. Contoh *Grain* Invoice

```
1 baris = 1 invoice
```

Cocok untuk laporan keuangan, piutang, status pembayaran, dan audit.

Contoh *measure*:

- invoice_amount;
- paid_amount;
- outstanding_amount;
- tax_amount;
- overdue_days.

Dimensi yang relevan:

- *DimDate*;
- *DimCustomer*;
- *DimRegion*;
- *DimPaymentStatus*;
- *DimSalesChannel*.

C. Contoh *Grain* Proyek

```
1 baris = 1 progres proyek per minggu
```

Cocok untuk konstruksi, infrastruktur, proyek teknologi, atau proyek operasional.

Contoh *measure*:

- planned_progress_percent;
- actual_progress_percent;
- progress_variance;
- planned_cost;
- actual_cost;
- cost_variance.

Dimensi yang relevan:

- *DimProject*;
- *DimDate*;
- *DimLocation*;
- *DimVendor*;
- *DimWorkPackage*;
- *DimProjectManager*.

D. Contoh *Grain* Energi

```
1 baris = 1 pemakaian energi per jam
```

Cocok untuk *green data warehouse*, efisiensi energi, operasi fasilitas, dan perhitungan biaya energi.

Contoh *measure*:

- kWh;
- peak_load;
- power_factor;
- cost_energy;
- emission_estimate.

Dimensi yang relevan:

- *DimDate*;
- *DimTime*;
- *DimLocation*;
- *DimDevice*;
- *DimEnergySource*.

E. Checklist Menentukan *Grain*

Gunakan pertanyaan berikut:

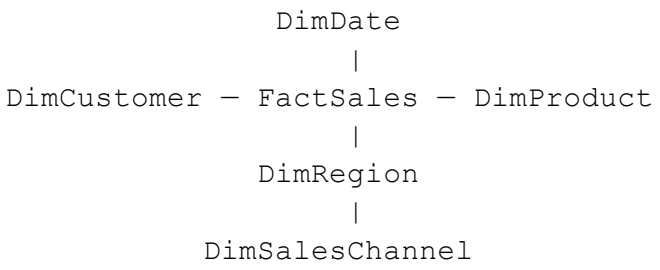
Pertanyaan	Jawaban yang Harus Jelas
Satu baris mewakili apa?	transaksi, item, invoice, minggu proyek, jam energi
Apakah data dapat dijumlahkan?	ya / tidak / terbatas
Apakah ada risiko duplikasi?	identifikasi kunci unik
Dimensi apa yang melekat pada baris?	waktu, pelanggan, produk, lokasi
Apakah histori dibutuhkan?	ya / tidak
Apakah detail cukup untuk kebutuhan analitik?	ya / tidak
Apakah volume data masih masuk akal?	estimasi baris per bulan / tahun

Prinsip kritis: pilih *grain* paling detail yang masih realistis secara biaya, performa, dan kebutuhan bisnis.

5.3 *Star Schema*

Star schema adalah model dimensional dengan satu *fact table* di tengah dan beberapa *dimension table* di sekelilingnya. Disebut “star” karena bentuk diagramnya seperti bintang.

Contoh:



Referensi yang digunakan menjelaskan bahwa *star schema* memiliki *fact table* pusat yang dikelilingi *dimension table*. Model ini sederhana, mudah dipahami, dan umum digunakan untuk *query* analitik.

A. Struktur *Star Schema*

Contoh *FactSales*:

Kolom	Jenis
sales_key	primary key teknis / opsional
date_key	foreign key
customer_key	foreign key
product_key	foreign key
region_key	foreign key
sales_channel_key	foreign key
quantity	measure
gross_amount	measure
discount_amount	measure
net_amount	measure
margin_amount	measure

Contoh *DimCustomer*:

Kolom	Jenis
customer_key	surrogate key
customer_id	natural key
customer_name	atribut
customer_type	atribut
city	atribut
province	atribut
region	atribut
active_status	atribut

B. Kelebihan *Star Schema*

1. Sederhana

Pengguna bisnis mudah memahami hubungan antar tabel. *Fact table* menyimpan angka, dimensi menyimpan konteks.

2. Cepat untuk *BI*

Karena dimensi cenderung *denormalized*, *query* membutuhkan lebih sedikit *join*. Ini penting untuk *dashboard* yang sering diakses.

3. Cocok untuk sertifikasi dan praktik profesional

Sebagian besar konsep dasar *Data Warehouse* dapat diajarkan melalui *star schema*: *fact*, *dimension*, *measure*, *grain*, *surrogate key*, dan *hierarchy*.

4. Mudah diperluas

Dimensi baru dapat ditambahkan tanpa merombak total model.

C. Kekurangan *Star Schema*

1. Redundansi data

Karena dimensi tidak dinormalisasi penuh, beberapa nilai bisa berulang.

Contoh:

product_name	category	division
Produk A	Elektronik	Retail
Produk B	Elektronik	Retail
Produk C	Elektronik	Retail

Kategori dan divisi berulang.

2. Risiko inkonsistensi jika governance lemah

Jika data dimensi tidak dikelola baik, nilai kategori, wilayah, atau status bisa tidak konsisten.

3. Storage lebih besar dibanding model normalisasi

Redundansi dimensi dapat meningkatkan kebutuhan storage. Namun dalam banyak kasus analitik, biaya storage lebih kecil daripada biaya *query* lambat.

D. Kapan Memakai *Star Schema*?

Gunakan *star schema* ketika:

- kebutuhan utama adalah *BI dashboard*;
- pengguna bisnis harus mudah memahami model;
- performa *query* penting;
- struktur dimensi tidak terlalu kompleks;
- laporan banyak melakukan agregasi;
- organisasi baru mulai membangun *Data Warehouse*.

Rekomendasi praktis: untuk sertifikasi dan implementasi awal, gunakan *star schema* sebagai default. Pindah ke *snowflake schema* hanya jika ada alasan kuat.

5.4 *Snowflake Schema*

Snowflake schema adalah pengembangan dari *star schema* dengan menormalisasi beberapa *dimension table* menjadi tabel-tabel turunan. Bentuknya lebih bercabang, seperti serpihan salju.

Contoh:

```
FactSales
   |
DimProduct
   |
DimSubCategory
   |
DimCategory
```

Referensi yang digunakan menjelaskan bahwa *snowflake schema* menormalisasi tabel dimensi untuk mengurangi redudansi, tetapi membuat struktur lebih kompleks dan dapat menambah jumlah *join*.

A. Contoh *Snowflake Schema*

Dalam *star schema*, *DimProduct* bisa berisi:

product_key	product_name	subcategory	category	division
1	Laptop A	Laptop	Elektronik	Retail
2	Laptop B	Laptop	Elektronik	Retail
3	Kamera A	Kamera	Elektronik	Retail

Dalam *snowflake schema*, struktur dipecah:

```
DimProduct
DimSubCategory
DimCategory
DimDivision
```

Contoh:

Tabel	Isi
DimProduct	product_key, product_name, subcategory_key
DimSubCategory	subcategory_key, subcategory_name, category_key
DimCategory	category_key, category_name, division_key
DimDivision	division_key, division_name

B. Kelebihan *Snowflake Schema*

1. Redundansi lebih rendah

Data kategori, wilayah, atau hierarki besar tidak diulang berkali-kali.

2. Integritas data lebih baik

Karena tabel dinormalisasi, perubahan atribut tertentu dapat lebih terkendali.

3. Cocok untuk hierarki kompleks

Misalnya:

- wilayah nasional → provinsi → kota → cabang;
- produk → subkategori → kategori → divisi;
- organisasi → divisi → departemen → unit;
- proyek → program → portofolio.

4. Lebih hemat storage pada dimensi besar

Jika dimensi sangat besar dan banyak atribut berulang, normalisasi dapat membantu.

C. Kekurangan *Snowflake Schema*

1. *Query* lebih kompleks

Pengguna atau alat analitik perlu melakukan lebih banyak *join*.

2. Performa bisa turun

Jika *join* terlalu banyak, *query* dashboard bisa melambat.

3. Lebih sulit dipahami pengguna bisnis

Model bercabang lebih sulit dibaca dibanding *star schema*.

4. Maintenance analitik lebih rumit

Perubahan kecil di satu tabel dapat mempengaruhi banyak relasi.

Referensi lain juga menekankan bahwa *snowflake schema* dapat mengurangi redudansi dan meningkatkan integritas data, tetapi menambah kompleksitas *query* dan potensi overhead performa.

D. Kapan Memakai *Snowflake Schema*?

Gunakan *snowflake schema* ketika:

- dimensi memiliki hierarki sangat besar;
- redundansi dimensi menjadi masalah nyata;
- integritas referensi lebih penting daripada kesederhanaan;
- data dimensi sering dipakai lintas model;
- organisasi punya tim data yang cukup matang;
- kebutuhan audit struktur data sangat kuat.

Prinsip kritis: jangan melakukan *snowflaking* hanya karena terlihat lebih “rapi”. Dalam analitik, model paling berguna adalah model yang **cepat, jelas, dan dipercaya**.

5.5 *Slowly Changing Dimension*

Slowly Changing Dimension atau SCD adalah teknik untuk menangani perubahan atribut dimensi dari waktu ke waktu.

Contoh perubahan:

- pelanggan pindah kota;
- produk berubah kategori;
- cabang berubah region;
- vendor berubah status;
- pegawai pindah departemen;
- proyek berubah manajer.

Pertanyaannya: ketika atribut berubah, apakah histori lama harus ikut berubah atau tetap disimpan?

Kimball Group menjelaskan teknik SCD seperti Type 0, Type 1, Type 2, Type 3, dan beberapa variasi lanjutan. Type 0 mempertahankan nilai asli, Type 1 menimpa nilai lama, Type 2 menambah baris baru, dan Type 3 menambah atribut baru untuk menyimpan histori terbatas. (Kimball Group)

A. SCD Type 0 — Nilai Tidak Diubah

Type 0 berarti nilai atribut tidak berubah. Nilai pertama dipertahankan.

Cocok untuk:

- tanggal lahir;
- tanggal registrasi;
- kode awal pelanggan;
- asal pelanggan pertama;
- tanggal mulai proyek.

Contoh:

customer_key	customer_id	registration_date
1001	C001	2022-01-15

Jika ada koreksi, biasanya harus melalui proses audit, bukan update biasa.

Gunakan Type 0 jika nilai historis asli harus dipertahankan.

B. SCD Type 1 — *Overwrite*

Type 1 menimpa nilai lama dengan nilai baru. Histori lama tidak disimpan.

Contoh sebelum:

customer_key	customer_id	customer_name	city
1001	C001	PT Abadi	Jakarta

Setelah koreksi nama:

customer_key	customer_id	customer_name	city
1001	C001	PT Abadi Jaya	Jakarta

Cocok untuk:

- koreksi ejaan;
- koreksi kesalahan input;
- perubahan yang tidak penting secara historis;
- atribut yang tidak dipakai untuk analisis histori.

Kimball menjelaskan bahwa Type 1 menimpa informasi yang berubah pada dimensi. (Kimball Group)

Gunakan Type 1 jika histori lama tidak penting atau memang salah.

C. SCD Type 2 — Baris Baru untuk Histori

Type 2 menambah baris baru ketika atribut berubah. Setiap versi memiliki *surrogate key* berbeda.

Contoh pelanggan pindah kota:

customer_key	customer_id	customer_name	city	start_date	end_date	current_flag
1001	C001	PT Abadi Jaya	Jakarta	2022-01-01	2024-12-31	N
1450	C001	PT Abadi Jaya	Bandung	2025-01-01	9999-12-31	Y

Manfaat:

- history tetap tersimpan;
- analisis masa lalu tetap benar;
- fakta lama dapat terhubung ke versi dimensi lama;
- cocok untuk audit.

Cocok untuk:

- kota pelanggan;
- segmentasi pelanggan;
- status vendor;
- region cabang;
- kategori produk;
- departemen pegawai;
- manajer proyek.

Wikipedia menjelaskan bahwa Type 2 melacak histori dengan membuat banyak record untuk satu *natural key*, biasanya dengan *surrogate key* berbeda dan tanggal berlaku. (Wikipedia)

Gunakan Type 2 jika perubahan atribut mempengaruhi analisis historis.

D. SCD Type 3 — Kolom Tambahan untuk Histori Terbatas

Type 3 menyimpan histori terbatas dalam kolom tambahan.

Contoh:

customer_key	customer_id	customer_name	previous_city	current_city
1001	C001	PT Abadi Jaya	Jakarta	Bandung

Cocok untuk:

- menyimpan nilai lama terbatas;
- analisis sebelum-sesudah sederhana;
- perubahan yang jarang;
- kebutuhan bisnis hanya perlu nilai sebelumnya dan saat ini.

Kekurangan:

- tidak cocok untuk histori panjang;
- struktur tabel perlu berubah jika ingin menyimpan lebih banyak versi;
- tidak sekuat Type 2 untuk audit.

Wikipedia menjelaskan bahwa Type 3 menambahkan kolom tambahan untuk menyimpan histori terbatas, misalnya nilai sebelumnya dan nilai saat ini. (Wikipedia)

Gunakan Type 3 jika hanya perlu membandingkan kondisi sebelumnya dan sekarang.

E. Ringkasan Pemilihan SCD

Tipe SCD	Cara Kerja	Histori	Cocok Untuk
Type 0	tidak berubah	nilai asli	tanggal lahir, tanggal registrasi
Type 1	timpa nilai lama	tidak ada	koreksi ejaan, data salah
Type 2	tambah baris baru	lengkap	segmentasi, wilayah, status
Type 3	tambah kolom	terbatas	nilai lama vs nilai baru

Prinsip kritis: pilihan SCD bukan keputusan teknis semata. Ini keputusan bisnis dan audit: apakah perubahan lama harus tetap terlihat?

5.6 Contoh Desain: *FactSales* dan Dimensi Penjualan

Bagian ini membuat desain sederhana untuk *Data Warehouse* penjualan.

Tujuan analitik:

- melihat penjualan per hari, bulan, dan tahun;
- melihat penjualan per produk;
- melihat penjualan per pelanggan;
- melihat performa wilayah;
- melihat kontribusi kanal penjualan;
- menghitung margin.

A. Menentukan *Grain*

Untuk contoh ini:

```
1 baris FactSales = 1 item transaksi penjualan
```

Artinya, jika satu transaksi memiliki tiga produk, maka akan ada tiga baris di *FactSales*.

Alasan:

- analisis produk lebih akurat;
- margin per produk bisa dihitung;
- kuantitas per produk jelas;
- agregasi tetap bisa naik ke level transaksi, hari, bulan, cabang, atau wilayah.

B. Tabel *FactSales*

Kolom	Tipe	Keterangan
sales_fact_key	bigint	kunci teknis
sales_id	varchar	ID transaksi dari sumber
line_number	integer	nomor baris item
date_key	integer	FK ke DimDate
customer_key	integer	FK ke DimCustomer
product_key	integer	FK ke DimProduct
region_key	integer	FK ke DimRegion
sales_channel_key	integer	FK ke DimSalesChannel
quantity	numeric	jumlah unit
gross_amount	numeric	nilai kotor
discount_amount	numeric	diskon
net_amount	numeric	nilai bersih
cost_amount	numeric	biaya pokok
margin_amount	numeric	margin

Rumus dasar:

```
net_amount = gross_amount - discount_amount
margin_amount = net_amount - cost_amount
```

C. DimCustomer

Kolom	Keterangan
customer_key	surrogate key
customer_id	kode pelanggan dari sumber
customer_name	nama pelanggan
customer_type	tipe pelanggan
segment	segmentasi
city	kota
province	provinsi
region	wilayah
start_date	awal versi
end_date	akhir versi
current_flag	penanda versi aktif

Rekomendasi SCD:

- customer_name: Type 1 jika hanya koreksi ejaan;
- city / region / segment: Type 2 jika penting untuk analisis histori;
- registration_date: Type 0.

D. *DimProduct*

Kolom	Keterangan
product_key	surrogate key
product_id	kode produk
product_name	nama produk
category	kategori
subcategory	subkategori
brand	merek
unit	satuan
product_status	aktif / tidak aktif

Rekomendasi:

- jika kategori produk sering berubah dan analisis histori penting, gunakan Type 2;
- jika hanya koreksi nama, gunakan Type 1;
- jika hierarki produk sangat kompleks, pertimbangkan *snowflake schema*.

E. *DimDate*

Kolom	Keterangan
date_key	format YYYYMMDD
full_date	tanggal lengkap
day	hari
month	bulan
month_name	nama bulan
quarter	kuartal
year	tahun
week_of_year	minggu ke-
is_weekend	akhir pekan / bukan

DimDate hampir selalu diperlukan dalam *Data Warehouse* karena hampir semua analisis membutuhkan waktu.

F. *DimRegion*

Kolom	Keterangan
region_key	surrogate key
branch_code	kode cabang
branch_name	nama cabang
city	kota
province	provinsi
region_name	nama region
country	negara

Jika struktur wilayah kompleks, *DimRegion* bisa di-*snowflake* menjadi:

```
DimBranch → DimCity → DimProvince → DimRegion
```

Namun untuk kebutuhan awal *BI*, satu tabel *DimRegion* sering lebih praktis.

G. *DimSalesChannel*

Kolom	Keterangan
sales_channel_key	surrogate key
channel_code	kode kanal
channel_name	nama kanal
channel_type	offline / online / partner
channel_group	kelompok kanal

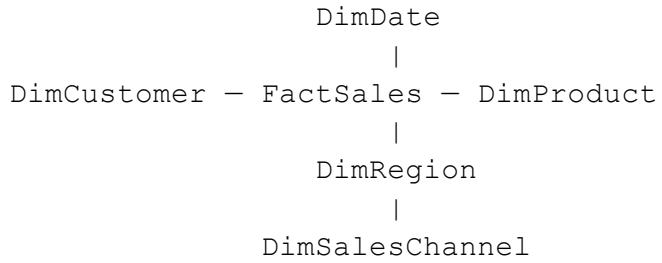
Contoh nilai:

- cabang;
- agen;
- distributor;
- aplikasi;
- mitra;
- layanan langsung.

Hands-on. Membuat Star Schema untuk Data Penjualan

Latihan ini menggunakan pendekatan SQL relasional yang dapat dijalankan pada PostgreSQL atau MariaDB dengan penyesuaian kecil. Fokus latihan adalah memahami struktur, bukan optimasi lanjutan.

Langkah 1 — Gambar Model



Langkah 2 — Buat Tabel Dimensi

DimDate

```
CREATE TABLE dim_date (
    date_key        INTEGER PRIMARY KEY,
    full_date       DATE NOT NULL,
    day_number      INTEGER NOT NULL,
    month_number    INTEGER NOT NULL,
    month_name      VARCHAR(20) NOT NULL,
    quarter_number  INTEGER NOT NULL,
    year_number     INTEGER NOT NULL,
    is_weekend      BOOLEAN NOT NULL
);
```

DimCustomer

```
CREATE TABLE dim_customer (
    customer_key    INTEGER PRIMARY KEY,
    customer_id     VARCHAR(50) NOT NULL,
    customer_name   VARCHAR(200) NOT NULL,
    customer_type   VARCHAR(50),
    segment         VARCHAR(50),
    city            VARCHAR(100),
    province        VARCHAR(100),
    region_name     VARCHAR(100),
    start_date      DATE NOT NULL,
    end_date        DATE NOT NULL,
    current_flag    CHAR(1) NOT NULL
);
```

DimProduct

```
CREATE TABLE dim_product (
    product_key     INTEGER PRIMARY KEY,
    product_id      VARCHAR(50) NOT NULL,
    product_name    VARCHAR(200) NOT NULL,
    category        VARCHAR(100),
    subcategory     VARCHAR(100),
    brand           VARCHAR(100),
    unit_name       VARCHAR(50),
    product_status  VARCHAR(50)
);
```

DimRegion

```
CREATE TABLE dim_region (
    region_key      INTEGER PRIMARY KEY,
    branch_code     VARCHAR(50) NOT NULL,
    branch_name     VARCHAR(200),
    city            VARCHAR(100),
    province        VARCHAR(100),
    region_name     VARCHAR(100),
    country_name    VARCHAR(100)
);
```

DimSalesChannel

```
CREATE TABLE dim_sales_channel (
    sales_channel_key INTEGER PRIMARY KEY,
    channel_code      VARCHAR(50) NOT NULL,
    channel_name      VARCHAR(100),
    channel_type      VARCHAR(50),
    channel_group     VARCHAR(100)
);
```

Langkah 3 — Buat *FactSales*

```
CREATE TABLE fact_sales (
    sales_fact_key     BIGINT PRIMARY KEY,
    sales_id           VARCHAR(50) NOT NULL,
    line_number        INTEGER NOT NULL,

    date_key           INTEGER NOT NULL,
    customer_key       INTEGER NOT NULL,
    product_key        INTEGER NOT NULL,
    region_key         INTEGER NOT NULL,
    sales_channel_key  INTEGER NOT NULL,

    quantity           NUMERIC(18,2) NOT NULL,
    gross_amount       NUMERIC(18,2) NOT NULL,
    discount_amount    NUMERIC(18,2) NOT NULL,
    net_amount         NUMERIC(18,2) NOT NULL,
    cost_amount        NUMERIC(18,2) NOT NULL,
    margin_amount      NUMERIC(18,2) NOT NULL,

    FOREIGN KEY (date_key) REFERENCES dim_date(date_key),
    FOREIGN KEY (customer_key) REFERENCES
dim_customer(customer_key),
    FOREIGN KEY (product_key) REFERENCES
dim_product(product_key),
    FOREIGN KEY (region_key) REFERENCES
dim_region(region_key),
    FOREIGN KEY (sales_channel_key) REFERENCES
dim_sales_channel(sales_channel_key)
);
```

Relasi *foreign key* seperti di atas penting untuk menjaga integritas antara *fact table* dan *dimension table*. Dokumentasi PostgreSQL menjelaskan bahwa *foreign key constraint* memastikan nilai pada kolom anak sesuai dengan baris yang ada di tabel referensi. (PostgreSQL)

Langkah 4 — Masukkan Contoh Data Dimensi

```
INSERT INTO dim_date VALUES
(20260101, '2026-01-01', 1, 1, 'Januari', 1, 2026, false),
(20260102, '2026-01-02', 2, 1, 'Januari', 1, 2026, false);

INSERT INTO dim_customer VALUES
(1, 'C001', 'PT Nusantara Maju', 'Corporate', 'Gold',
 'Jakarta', 'DKI Jakarta', 'Region Barat',
 '2026-01-01', '9999-12-31', 'Y');

INSERT INTO dim_product VALUES
(1, 'P001', 'Produk A', 'Elektronik', 'Perangkat', 'Brand
Lokal', 'unit', 'active'),
(2, 'P002', 'Produk B', 'Elektronik', 'Aksesori', 'Brand
Lokal', 'unit', 'active');

INSERT INTO dim_region VALUES
(1, 'JKT01', 'Cabang Jakarta 1', 'Jakarta', 'DKI Jakarta',
'Region Barat', 'Indonesia');

INSERT INTO dim_sales_channel VALUES
(1, 'OFF01', 'Cabang', 'offline', 'direct');
```

Langkah 5 — Masukkan Contoh Data *Fact*

```
INSERT INTO fact_sales VALUES
(1, 'S001', 1, 20260101, 1, 1, 1, 1, 2, 2000000, 100000,
1900000, 1400000, 500000),
(2, 'S001', 2, 20260101, 1, 2, 1, 1, 1, 500000, 0, 500000,
300000, 200000);
```

Langkah 6 — Query Analitik Dasar

Penjualan bersih per bulan

```
SELECT
    d.year_number,
    d.month_number,
    SUM(f.net_amount) AS total_net_sales
FROM fact_sales f
JOIN dim_date d ON f.date_key = d.date_key
GROUP BY d.year_number, d.month_number
ORDER BY d.year_number, d.month_number;
```

Penjualan per produk

```
SELECT
    p.category,
    p.product_name,
    SUM(f.quantity) AS total_quantity,
    SUM(f.net_amount) AS total_sales,
    SUM(f.margin_amount) AS total_margin
FROM fact_sales f
JOIN dim_product p ON f.product_key = p.product_key
GROUP BY p.category, p.product_name
ORDER BY total_sales DESC;
```

Penjualan per region

```
SELECT
    r.region_name,
    r.province,
    SUM(f.net_amount) AS total_sales
FROM fact_sales f
JOIN dim_region r ON f.region_key = r.region_key
GROUP BY r.region_name, r.province
ORDER BY total_sales DESC;
```

Margin per kanal penjualan

```
SELECT
    c.channel_group,
    c.channel_name,
    SUM(f.net_amount) AS total_sales,
    SUM(f.margin_amount) AS total_margin,
    ROUND(SUM(f.margin_amount) / NULLIF(SUM(f.net_amount), 0)
* 100, 2) AS margin_percent
FROM fact_sales f
JOIN dim_sales_channel c ON f.sales_channel_key =
c.sales_channel_key
GROUP BY c.channel_group, c.channel_name
ORDER BY margin_percent DESC;
```

Langkah 7 — Validasi Kualitas Model

Gunakan checklist berikut:

Pemeriksaan	Tujuan
Apakah grain sudah jelas?	mencegah hitung ganda
Apakah semua FK valid?	menjaga integritas referensial
Apakah measure bisa dijumlahkan?	memastikan agregasi benar
Apakah dimensi cukup deskriptif?	memudahkan analisis
Apakah histori pelanggan diperlukan?	menentukan SCD
Apakah DimDate lengkap?	mendukung analisis waktu
Apakah produk dan region konsisten?	mencegah KPI salah
Apakah data sensitif terlindungi?	menjaga compliance

Kesalahan Umum dalam *Dimensional Modeling*

Kesalahan yang sering terjadi:

- membuat *fact table* tanpa *grain* yang jelas;
- memasukkan atribut deskriptif terlalu banyak ke *fact table*;
- tidak memakai *surrogate key*;
- memakai kode sumber langsung sebagai kunci utama analitik;
- mencampur beberapa *grain* dalam satu *fact table*;
- membuat dimensi terlalu teknis dan tidak dipahami bisnis;
- semua perubahan dimensi ditimpa tanpa mempertimbangkan histori;
- memakai *snowflake schema* tanpa kebutuhan nyata;
- tidak membuat *DimDate*;
- tidak mendokumentasikan definisi *measure*.

Prinsip kritis: model dimensional yang buruk membuat dashboard terlihat benar, tetapi angka di dalamnya salah secara konseptual.

Checklist Sertifikasi Bab 5

Kompetensi	Harus Bisa
Fact table	menjelaskan fungsi dan contoh
Dimension table	menjelaskan atribut kontekstual
Measure	membedakan angka KPI dan atribut
Grain	menentukan satu baris mewakili apa
Hierarchy	membuat level analisis
Surrogate key	menjelaskan fungsi dan manfaat
Star schema	membuat model sederhana
Snowflake schema	menjelaskan kapan digunakan
SCD Type 0–3	memilih strategi histori dimensi
SQL dasar	membuat tabel dan query analitik

Ringkasan Bab

Dimensional modeling adalah fondasi desain analitik dalam *Data Warehouse*. Model ini menyusun data ke dalam *fact table* dan *dimension table* agar pengguna dapat melakukan analisis dengan cepat, jelas, dan konsisten. Referensi yang digunakan menekankan bahwa *star schema* sederhana dan cepat untuk *query*, sedangkan *snowflake schema* mengurangi redundansi tetapi menambah kompleksitas.

Poin penting:

- **Mulai dari proses bisnis dan grain.**
- *Fact table* menyimpan kejadian dan angka yang diukur.
- *Dimension table* menyimpan konteks bisnis.
- *Measure* adalah angka yang dianalisis.
- *Grain* menentukan tingkat detail satu baris fakta.
- *Star schema* cocok untuk *BI*, sederhana, dan cepat dipahami.
- *Snowflake schema* cocok untuk hierarki kompleks dan pengurangan redundansi.
- SCD digunakan untuk mengelola perubahan atribut dimensi dari waktu ke waktu.
- *Surrogate key* penting untuk integrasi dan histori.
- Model yang baik harus mudah dipahami, cepat di-*query*, dan dapat diaudit.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Jelaskan perbedaan *fact table* dan *dimension table*.
2. Mengapa *grain* harus ditentukan sebelum membuat tabel?
3. Apa kelebihan utama *star schema* untuk *BI dashboard*?
4. Kapan *snowflake schema* lebih tepat digunakan?
5. Jelaskan perbedaan SCD Type 0, Type 1, Type 2, dan Type 3.
6. Mengapa *surrogate key* penting dalam *Data Warehouse*?
7. Apa risiko jika satu *fact table* mencampur beberapa *grain*?
8. Mengapa *DimDate* hampir selalu diperlukan?

B. Studi Kasus

Sebuah perusahaan ingin membuat *Data Warehouse* penjualan nasional. Satu transaksi dapat berisi banyak produk. Direksi ingin melihat penjualan per bulan, produk, wilayah, pelanggan, dan kanal penjualan.

Tugas:

- tentukan *grain* yang tepat;
- buat daftar *measure*;
- buat daftar dimensi;
- tentukan atribut utama setiap dimensi;
- pilih *star schema* atau *snowflake schema*;
- tentukan minimal dua atribut yang perlu SCD Type 2;
- jelaskan alasannya.

C. Praktik

Buat rancangan *star schema* untuk data penjualan dengan struktur berikut:

Komponen	Isi
Nama fact table	
Grain	
Measure	
Dimensi waktu	
Dimensi pelanggan	
Dimensi produk	
Dimensi wilayah	
Dimensi kanal	
Strategi SCD	
Risiko kualitas data	

BAB 6. *ETL* dan *ELT*: *Extract, Transform, Load* untuk Profesional

Data Warehouse tidak akan bernilai jika data tidak dapat masuk secara benar, teratur, dan dapat dipercaya. Di sinilah *ETL* dan *ELT* menjadi inti operasional *Data Warehouse*. Keduanya bukan sekadar proses memindahkan data. Keduanya adalah **mesin integrasi data** yang menentukan apakah laporan, *dashboard*, audit, dan analitik akan menghasilkan angka yang benar.

Dalam referensi yang digunakan, *ETL* dijelaskan sebagai proses untuk mengekstrak data dari berbagai sumber, mentransformasikannya agar sesuai dengan kebutuhan analitik, lalu memuatnya ke *Data Warehouse*. Proses ini mencakup *cleansing*, standardisasi, integrasi, validasi, dan pemuatan data. IBM juga menjelaskan bahwa *ETL* dan *ELT* sama-sama memindahkan data mentah dari sistem sumber ke target seperti *data warehouse* atau *data lake*, dengan perbedaan utama pada urutan transformasi. (IBM)

Golden Rule: *Pipeline* data yang baik bukan hanya berhasil jalan, tetapi **bisa diuji, bisa diaudit, bisa diulang, dan bisa dijelaskan saat terjadi masalah**.

6.1 Perbedaan *ETL* dan *ELT*

ETL dan *ELT* sama-sama digunakan untuk integrasi data. Perbedaannya ada pada **urutan proses transformasi**.

```
ETL = Extract → Transform → Load
ELT = Extract → Load → Transform
```

Dalam *ETL*, data diekstrak dari sumber, ditransformasikan di area pemrosesan atau *staging*, lalu dimuat ke *Data Warehouse*. Dalam *ELT*, data diekstrak, dimuat terlebih dahulu ke platform target, lalu transformasi dilakukan di dalam platform tersebut. IBM menyebut perbedaan paling jelas antara *ETL* dan *ELT* adalah urutan operasinya: *ELT* memuat data mentah ke target terlebih dahulu, kemudian mentransformasikannya saat dibutuhkan. (IBM)

A. *ETL*: Transformasi Sebelum Masuk *Data Warehouse*

Pada pendekatan *ETL*, data tidak langsung masuk ke *Data Warehouse Core*. Data terlebih dahulu melewati proses pembersihan, validasi, standardisasi, dan transformasi.

Alur:

```
Source System → Extract → Staging / Processing → Transform → Data Warehouse
```

Cocok ketika:

- data sumber **kotor dan tidak konsisten**;
- data sensitif harus di masking sebelum masuk;
- aturan bisnis sudah jelas;
- kualitas data harus dikontrol ketat;
- organisasi membutuhkan validasi sebelum data masuk *warehouse*;
- kapasitas transformasi di target terbatas;
- audit membutuhkan data yang sudah lolos kontrol.

Contoh:

- data pelanggan dari banyak cabang perlu deduplikasi;
- format tanggal berbeda antar sumber;
- kode produk harus distandarisasi;
- data personal harus di masking;
- transaksi negatif harus divalidasi apakah retur atau salah input.

Referensi yang digunakan menjelaskan bahwa *ETL tools* membantu ekstraksi dari banyak sumber, *data cleansing*, reorganisasi, transformasi, agregasi, kalkulasi, dan pemuatan otomatis ke sistem tujuan.

B. *ELT*: *Load* Dulu, Transformasi di Platform Target

Pada pendekatan *ELT*, data mentah dimuat terlebih dahulu ke target. Transformasi dilakukan setelah data berada di platform tujuan.

Alur:

```
Source System → Extract → Load Raw Data → Transform in Target Platform
```

Cocok ketika:

- volume data besar;
- platform target kuat untuk komputasi;
- kebutuhan transformasi sering berubah;
- data mentah perlu disimpan untuk audit;
- eksplorasi data diperlukan sebelum model final;
- organisasi ingin fleksibilitas tinggi;
- pendekatan *data lake* atau *lakehouse* digunakan.

Referensi yang digunakan menjelaskan bahwa *ELT* memisahkan fase *extract-load* dari fase transformasi sehingga proses dapat dipecah menjadi bagian yang lebih kecil, lebih mudah dikelola, dan lebih fleksibel untuk kebutuhan masa depan. Referensi lain juga menyebut bahwa pendekatan *ELT* dapat memanfaatkan kekuatan komputasi *Data Warehouse* modern untuk melakukan transformasi setelah data dimuat.

C. Ringkasan Perbandingan *ETL* dan *ELT*

Aspek	**ETL**	**ELT**
Urutan proses	extract → transform → load	extract → load → transform
Data masuk target	sudah dibersihkan	bisa masih mentah
Cocok untuk	kontrol ketat, data sensitif, aturan stabil	data besar, eksplorasi, transformasi fleksibel
Kelebihan	kualitas lebih terkendali sebelum masuk DW	fleksibel, cepat menyimpan data mentah
Kekurangan	kurang fleksibel jika kebutuhan berubah	risiko data mentah menumpuk tanpa governance
Risiko utama	transformasi terlalu kaku	target menjadi “raw dump” tanpa kontrol
Kebutuhan governance	tinggi	sangat tinggi

Prinsip kritis: *ETL* lebih disiplin di depan; *ELT* lebih fleksibel di belakang. Keduanya baik jika desain, kualitas data, metadata, dan audit dibuat serius.

D. Kapan *ETL* Lebih Cocok?

Gunakan *ETL* jika:

- data harus bersih sebelum masuk *warehouse*;
- data sensitif tidak boleh masuk mentah;
- aturan bisnis stabil;
- sistem target tidak dirancang untuk transformasi besar;
- organisasi membutuhkan kontrol kualitas sebelum *load*;
- pengguna bisnis hanya boleh melihat data yang sudah tervalidasi.

Contoh kasus:

- data keuangan;
- data audit;
- data personal;
- data transaksi resmi;
- data regulasi;
- data laporan direksi.

E. Kapan *ELT* Lebih Cocok?

Gunakan *ELT* jika:

- volume data sangat besar;
- transformasi sering berubah;
- data mentah perlu disimpan sebagai arsip;
- platform target kuat untuk komputasi;
- analitik eksploratif dibutuhkan;
- organisasi ingin memisahkan proses *load* dan proses transformasi.

Contoh kasus:

- log aplikasi;
- data sensor;
- data historis besar;
- data semi-terstruktur;
- data untuk eksperimen analitik;
- integrasi awal dari banyak sumber.

Catatan kritis: *ELT* bukan alasan untuk memasukkan semua data tanpa desain. Jika tidak ada kontrol, *ELT* hanya mempercepat terbentuknya **tempat sampah data yang mahal**.

6.2 *Extract*

Extract adalah proses mengambil data dari sistem sumber. Tahap ini tampak sederhana, tetapi sering menjadi sumber masalah besar. Jika ekstraksi salah, seluruh proses berikutnya akan salah.

Sumber data dapat berupa:

- sistem transaksi;
- database relasional;
- file CSV;
- API;
- log aplikasi;
- data eksternal;
- data historis;
- data sensor.

Referensi implementasi *Data Warehouse* menekankan bahwa sumber data bisa berasal dari sistem operasional, *external data feeds*, spreadsheet, dan *flat files*, lalu diproses melalui *ETL* sebelum masuk *warehouse*.

A. *Full Extract*

Full extract mengambil seluruh data dari sumber setiap kali proses berjalan.

Contoh:

> Ambil seluruh tabel pelanggan setiap malam.

Kelebihan:

- sederhana;
- mudah dipahami;
- cocok untuk tabel kecil;
- mudah dibandingkan ulang;
- cocok untuk *master data* kecil.

Kekurangan:

- berat untuk tabel besar;
- lambat;
- membebani sistem sumber;
- boros storage dan jaringan;
- sulit untuk data transaksi besar.

Cocok untuk:

- tabel referensi kecil;
- data cabang;
- data kategori produk;
- data status;
- data kalender;
- file kecil yang berubah total.

B. *Incremental Extract*

Incremental extract hanya mengambil data yang berubah sejak proses terakhir.

Contoh:

> Ambil transaksi dengan updated_at > waktu ekstraksi terakhir.

Kelebihan:

- lebih cepat;
- lebih hemat sumber daya;
- cocok untuk data besar;
- cocok untuk proses harian;
- mengurangi beban sistem sumber.

Kekurangan:

- perlu kolom penanda perubahan;
- risiko data terlewat jika timestamp salah;
- perlu mekanisme *watermark*;
- perlu log proses yang rapi.

Contoh *watermark*:

Nama Proses	Last Successful Extract
extract_sales	2026-01-15 23:59:59
extract_customer	2026-01-15 23:30:00

Prinsip penting:

Ambil data baru berdasarkan waktu sukses terakhir, bukan waktu mulai proses.

C. *Change Data Capture* / CDC

Change Data Capture menangkap perubahan data seperti *insert*, *update*, dan *delete* dari sistem sumber.

Cocok untuk:

- kebutuhan data hampir real-time;
- tabel transaksi besar;
- integrasi banyak aplikasi;
- audit perubahan;
- sistem yang tidak boleh dibebani query besar.

Debezium adalah platform *open source* untuk *change data capture* yang dapat membaca perubahan baris dalam tabel database sebagai aliran kejadian perubahan, sehingga aplikasi atau pipeline dapat merespons perubahan *insert*, *update*, dan *delete*. (Debezium) Dokumentasi Debezium menjelaskan bahwa setiap perubahan tingkat baris dicatat dalam *change event stream* dan dapat dibaca aplikasi sesuai urutan perubahan. (Debezium)

Kelebihan CDC:

- perubahan data lebih cepat terdeteksi;
- mengurangi kebutuhan *full scan*;
- cocok untuk sinkronisasi;
- mendukung arsitektur *event-driven*.

Kekurangan CDC:

- konfigurasi lebih kompleks;
- perlu pemahaman log database;
- perlu monitoring ketat;
- perlu penanganan *schema change*;
- perlu desain *replay* jika proses gagal.

D. *API Extraction*

API extraction mengambil data melalui antarmuka aplikasi.

Cocok untuk:

- aplikasi modern;
- layanan eksternal;
- data yang tidak bisa diakses langsung dari database;
- integrasi antar sistem;
- data yang disediakan melalui endpoint resmi.

Yang harus diperhatikan:

- autentikasi;
- batas akses;
- pagination;
- rate limit;
- format respons;
- error code;
- timeout;
- perubahan skema;
- dokumentasi API;
- audit akses.

Contoh risiko:

- API berubah tanpa pemberitahuan;
- data hanya tersedia per halaman;
- respons API tidak konsisten;
- batas akses membuat ekstraksi lambat;
- data historis tidak tersedia penuh.

E. *File-Based Ingestion*

File-based ingestion mengambil data dari file seperti CSV, TSV, JSON, atau file hasil ekspor sistem.

Cocok untuk:

- sistem lama;
- proses manual terkontrol;
- data bulanan;
- data dari unit eksternal;
- prototipe awal;
- data referensi.

PostgreSQL menyediakan perintah `COPY` untuk memindahkan data antara tabel dan file; `COPY FROM` memuat data dari file ke tabel, sedangkan `COPY TO` menyalin isi tabel ke file. (PostgreSQL)

Risiko file:

- nama kolom berubah;
- urutan kolom berubah;
- delimiter tidak konsisten;
- encoding bermasalah;
- file ganda;
- data tidak lengkap;
- tidak ada kontrol versi.

Kontrol minimal:

- nama file standar;
- struktur kolom tetap;
- validasi jumlah kolom;
- validasi jumlah baris;
- checksum;
- folder input dan archive;
- log proses;
- status berhasil/gagal.

6.3 *Transform*

Transform adalah proses mengubah data mentah menjadi data yang siap dianalisis. Tahap ini adalah tempat aturan bisnis diterapkan.

Transformasi mencakup:

- *cleansing*;
- standardisasi;
- deduplikasi;
- *join*;
- agregasi;
- aturan bisnis;
- *data masking*.

Referensi desain *Data Warehouse* menekankan bahwa proses *ETL* harus mendefinisikan strategi ekstraksi, aturan transformasi, mekanisme pemuatan, serta langkah kualitas dan *cleansing* agar hanya data reliabel dan akurat yang masuk ke *warehouse*.

A. *Cleansing*

Cleansing adalah pembersihan data dari kesalahan dasar.

Contoh:

Masalah	Tindakan
tanggal kosong	tolak atau isi berdasarkan aturan
nilai negatif tidak valid	kirim ke tabel error
spasi berlebih	trim
huruf tidak konsisten	standardisasi kapitalisasi
karakter aneh	bersihkan sesuai aturan
format angka salah	konversi format
email tidak valid	tandai sebagai invalid

Contoh logika:

```
TRIM(customer_name)
LOWER(email)
amount >= 0
transaction_date IS NOT NULL
```

Prinsip kritis: jangan membersihkan data tanpa aturan bisnis. Data yang terlihat “aneh” belum tentu salah; bisa jadi retur, koreksi, atau kasus khusus.

B. Standardisasi

Standardisasi menyamakan format dan nilai agar data bisa digabungkan.

Contoh:

Elemen	Standar
Tanggal	YYYY-MM-DD
Mata uang	IDR
Kode cabang	3–5 karakter baku
Status proyek	planned, active, delayed, closed
Nama provinsi	sesuai master referensi
Satuan	unit, kg, ton, liter

Contoh transformasi:

```
"Jakarta", "DKI Jakarta", "JKT" → DKI Jakarta
"aktif", "Active", "A" → active
"01/02/2026" → 2026-02-01
```

Standardisasi adalah fondasi integrasi. Tanpa standarisasi, laporan lintas sistem akan penuh pengecualian.

C. *Deduplication*

Deduplication menghapus atau menggabungkan data ganda.

Contoh duplikasi:

- pelanggan sama dengan ejaan berbeda;
- invoice sama masuk dua kali;
- produk sama punya kode berbeda;
- vendor sama punya nama legal berbeda;
- transaksi diunggah ulang.

Metode sederhana:

- exact match;
- match berdasarkan kombinasi kolom;
- fuzzy match;
- rule-based matching;
- master data mapping.

Contoh aturan:

```
invoice_number + vendor_id + invoice_date harus unik
customer_id harus unik
sales_id + line_number harus unik
```

Prinsip kritis: deduplikasi harus hati-hati. Menghapus data ganda yang sebenarnya berbeda bisa lebih berbahaya daripada membiarkan duplikasi.

D. *Join*

Join menggabungkan data dari beberapa sumber.

Contoh:

- transaksi + pelanggan;
- transaksi + produk;
- invoice + pembayaran;
- proyek + vendor;
- progres proyek + rencana proyek;
- biaya + kode akun.

Risiko *join*:

- kunci tidak cocok;
- data master tidak lengkap;
- satu transaksi menjadi banyak baris;
- agregasi menjadi ganda;
- tabel referensi tidak terbaru.

Kontrol penting:

- cek jumlah baris sebelum dan sesudah *join*;
- cek data yang tidak punya pasangan;
- cek duplikasi kunci;
- simpan hasil validasi.

Contoh validasi:

```
SELECT COUNT(*) FROM stg_sales;

SELECT COUNT(*)
FROM stg_sales s
LEFT JOIN dim_product p ON s.product_id = p.product_id
WHERE p.product_key IS NULL;
```

E. Agregasi

Agregasi meringkas data dari level detail ke level lebih tinggi.

Contoh:

- transaksi harian → penjualan bulanan;
- progres mingguan → progres bulanan;
- invoice detail → total invoice;
- log per event → jumlah akses per jam;
- pemakaian energi per menit → pemakaian per jam.

Contoh SQL:

```
SELECT
    date_key,
    product_key,
    SUM(quantity) AS total_quantity,
    SUM(net_amount) AS total_sales
FROM fact_sales
GROUP BY date_key, product_key;
```

Agregasi mempercepat *dashboard*, tetapi harus hati-hati agar tidak menghilangkan detail penting.

F. *Business Rule*

Business rule adalah aturan bisnis yang diterapkan dalam transformasi.

Contoh:

Aturan Bisnis	Implementasi
pelanggan aktif	transaksi dalam 12 bulan terakhir
proyek terlambat	actual_progress < planned_progress
penjualan bersih	gross - discount - return
margin	net_sales - cost
invoice overdue	due_date < today dan belum lunas
transaksi anomali	nilai > batas tertentu

Prinsip kritis: aturan bisnis harus disetujui *data owner*. Jangan biarkan aturan KPI hanya hidup di kode ETL.

G. *Data Masking*

Data masking menyamarkan data sensitif agar aman dipakai untuk analitik, pengujian, atau pelaporan terbatas.

Contoh data yang perlu di masking:

- nomor identitas;
- nomor telepon;
- alamat lengkap;
- email pribadi;
- nomor rekening;
- data pegawai;
- data pelanggan sensitif.

Contoh masking:

```
081234567890 → 0812******90
nama@email.com → n***@email.com
1234567890123456 → ************3456
```

Masking penting untuk mengurangi risiko kebocoran data, terutama jika data diakses banyak pengguna.

6.4 *Load*

Load adalah proses memuat data ke tabel target. Tabel target bisa berupa:

- tabel *staging*;
- tabel dimensi;
- tabel fakta;
- tabel agregat;
- *data mart*;
- tabel audit;
- tabel error.

A. *Initial Load*

Initial load adalah pemuatan awal, biasanya mengambil data historis dalam jumlah besar.

Contoh:

```
Load transaksi 5 tahun terakhir ke FactSales.
```

Tantangan:

- volume besar;
- proses lama;
- kualitas data historis buruk;
- format lama berbeda;
- kunci berubah;
- risiko duplikasi;
- perlu rekonsiliasi.

Strategi:

- load per periode;
- validasi per batch;
- simpan log;
- lakukan checksum;
- cek jumlah baris;
- bandingkan total nilai dengan sumber;
- pisahkan data error.

B. Incremental Load

Incremental load memuat data baru atau berubah setelah proses terakhir.

Contoh:

```
Load transaksi baru sejak ETL terakhir sukses.
```

Kelebihan:

- cepat;
- hemat sumber daya;
- cocok untuk operasi harian;
- mengurangi beban sistem.

Risiko:

- data terlewat;
- data ganda;
- perubahan terlambat;
- delete tidak tertangani;
- update lama tidak terbaca.

Kontrol:

- gunakan *watermark*;
- gunakan tabel audit;
- cek duplikasi;
- simpan status batch;
- gunakan transaksi database;
- siapkan proses ulang.

C. *Batch Load*

Batch load memuat data secara terjadwal.

Contoh jadwal:

Proses	Jadwal
Load transaksi	setiap malam
Load pelanggan	setiap malam
Load produk	mingguan
Load target	bulanan
Load data historis	sesuai proyek

Kelebihan:

- stabil;
- mudah dikontrol;
- cocok untuk laporan periodik;
- biaya lebih mudah diprediksi.

Kekurangan:

- data tidak selalu terbaru;
- tidak cocok untuk peringatan real-time;
- jika gagal, laporan bisa terlambat.

D. *Near-Real-Time Load*

Near-real-time load memuat data dengan jeda pendek, misalnya setiap beberapa menit.

Cocok untuk:

- monitoring transaksi;
- deteksi anomali;
- operasional layanan;
- sensor;
- log aplikasi;
- alert risiko.

Risiko:

- arsitektur lebih kompleks;
- monitoring harus kuat;
- data terlambat perlu ditangani;
- kualitas data sulit dicek mendalam;
- biaya bisa lebih tinggi.

Prinsip kritis: jangan memilih *near-real-time* hanya karena terdengar modern. Pilih jika keputusan bisnis memang membutuhkan kecepatan itu.

E. *Error Handling*

Error handling menentukan apa yang terjadi jika data gagal diproses.

Jenis error:

- format data salah;
- field wajib kosong;
- foreign key tidak ditemukan;
- duplikasi;
- tipe data salah;
- nilai di luar batas;
- koneksi gagal;
- file tidak ditemukan;
- API timeout;
- query gagal.

Strategi:

Error	Tindakan
data salah format	masuk tabel error
field wajib kosong	tolak baris
kode referensi tidak valid	karantina
koneksi gagal	retry terbatas
file tidak ditemukan	status gagal dan notifikasi
duplikasi	skip atau merge sesuai aturan
batch gagal	rollback

OWASP menekankan bahwa transaksi yang mengalami error harus memiliki jejak audit dan kontrol integritas, serta proses gagal harus ditangani secara aman. (OWASP)

6.5 *Logging* dan Audit *ETL*

Logging adalah bukti bahwa proses berjalan dengan benar. Tanpa log, tim hanya tahu "berhasil" atau "gagal", tetapi tidak tahu **mengapa, kapan, berapa data diproses, dan apakah hasilnya konsisten**.

Referensi implementasi *Data Warehouse* menekankan pentingnya rekonsiliasi untuk membandingkan data dari sumber dengan data di *warehouse*, memastikan akurasi, menemukan ketidaksesuaian, dan menjaga kepercayaan terhadap output analitik. OWASP Logging Cheat Sheet memberikan panduan untuk membangun mekanisme *logging* aplikasi, terutama terkait keamanan dan audit. (OWASP Cheat Sheet Series)

A. *Row Count*

Row count mencatat jumlah baris pada setiap tahap.

Contoh:

Tahap	Jumlah Baris
source	1.000.000
extracted	1.000.000
staging	999.980
rejected	20
loaded fact	999.980

Jika jumlah baris berubah, harus ada alasan yang jelas.

B. *Error Count*

Error count mencatat jumlah error per jenis.

Contoh:

Jenis Error	Jumlah
tanggal kosong	10
kode produk tidak valid	5
nilai negatif tidak sah	3
duplikasi	2

Manfaat:

- memantau kualitas sumber data;
- memberi umpan balik ke *data owner*;
- mendukung perbaikan sistem sumber;
- mencegah error berulang.

C. *Start Time* dan *End Time*

Waktu mulai dan selesai penting untuk mengukur performa.

Contoh:

Proses	Start Time	End Time	Durasi
extract_sales	01:00	01:15	15 menit
transform_sales	01:15	01:40	25 menit
load_fact_sales	01:40	01:55	15 menit

Jika durasi meningkat, mungkin ada:

- volume data bertambah;
- indeks bermasalah;
- query lambat;
- sistem sumber lambat;
- jaringan bermasalah;
- transformasi tidak efisien.

D. *Source Checksum*

Checksum membantu memastikan file atau data tidak berubah tanpa terdeteksi.

Contoh penggunaan:

- memastikan file CSV tidak rusak;
- memastikan file tidak diproses dua kali;
- membandingkan versi file;
- mendeteksi perubahan diam-diam.

Contoh kolom log:

File Name	Checksum	Status
sales_20260101.csv	a91f...	processed
sales_20260102.csv	b81c...	processed

E. *Load Status*

Status proses harus jelas.

Contoh status:

- running;
- success;
- failed;
- partially_success;
- skipped;
- rollback;
- retrying.

Contoh tabel log:

batch_id	process_name	status	row_in	row_out	error_count
20260101-001	load_sales	success	100000	99980	20
20260101-002	load_customer	failed	5000	0	120

F. *Data Reconciliation*

Data reconciliation membandingkan data sumber dan target.

Contoh rekonsiliasi:

Pemeriksaan	Sumber	Target	Status
jumlah transaksi	1.000.000	1.000.000	OK
total penjualan	25.000.000.000	25.000.000.000	OK
jumlah invoice	20.000	19.998	Selisih
total pembayaran	12.000.000.000	12.000.000.000	OK

Rekonsiliasi wajib untuk:

- laporan keuangan;
- transaksi penting;
- audit;
- KPI direksi;
- data sensitif;
- data lintas sistem.

Prinsip kritis: log bukan formalitas teknis. Log adalah **bukti profesional** bahwa data dapat dipercaya.

Hands-on. Contoh Alur *ETL* CSV → *Staging* → *Fact Table* Menggunakan SQL / Python

Latihan ini membuat alur sederhana:

```
sales.csv → stg_sales_raw → stg_sales_clean → fact_sales
```

Tujuan:

- membaca file CSV;
- memuat ke *staging*;
- membersihkan data;
- memvalidasi data;
- memuat ke *fact table*;
- mencatat log proses.

Struktur File CSV

Contoh file `sales.csv`:

```
sales_id,line_number,transaction_date,customer_id,product_id,
branch_code,quantity,gross_amount,discount_amount,cost_amount
S001,1,2026-01-01,C001,P001,JKT01,2,2000000,100000,1400000
S001,2,2026-01-01,C001,P002,JKT01,1,500000,0,300000
S002,1,2026-01-02,C002,P001,BDG01,1,1000000,50000,700000
```

Langkah 1 — Buat Tabel *Staging*

```
CREATE TABLE stg_sales_raw (
    sales_id          VARCHAR(50),
    line_number       INTEGER,
    transaction_date  VARCHAR(50),
    customer_id       VARCHAR(50),
    product_id        VARCHAR(50),
    branch_code       VARCHAR(50),
    quantity          NUMERIC(18,2),
    gross_amount      NUMERIC(18,2),
    discount_amount   NUMERIC(18,2),
    cost_amount       NUMERIC(18,2),
    source_file       VARCHAR(200),
    load_batch_id     VARCHAR(50),
    loaded_at         TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Catatan: tanggal masih disimpan sebagai teks di *raw staging* agar data mentah tidak langsung gagal saat formatnya bermasalah.

Langkah 2 — Buat Tabel Error

```
CREATE TABLE etl_error_log (
    error_id       BIGSERIAL PRIMARY KEY,
    batch_id       VARCHAR(50),
    process_name   VARCHAR(100),
    source_key     VARCHAR(200),
    error_type     VARCHAR(100),
    error_message  TEXT,
    error_time     TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Langkah 3 — Buat Tabel Log Proses

```
CREATE TABLE etl_batch_log (
    batch_id       VARCHAR(50) PRIMARY KEY,
    process_name   VARCHAR(100),
    source_file    VARCHAR(200),
    start_time     TIMESTAMP,
    end_time       TIMESTAMP,
    status         VARCHAR(50),
    row_in         INTEGER,
    row_out        INTEGER,
    error_count    INTEGER,
    notes          TEXT
);
```

Langkah 4 — Buat Tabel Bersih

```
CREATE TABLE stg_sales_clean (
    sales_id          VARCHAR(50),
    line_number       INTEGER,
    transaction_date  DATE,
    customer_id       VARCHAR(50),
    product_id        VARCHAR(50),
    branch_code       VARCHAR(50),
    quantity          NUMERIC(18,2),
    gross_amount      NUMERIC(18,2),
    discount_amount   NUMERIC(18,2),
    net_amount        NUMERIC(18,2),
    cost_amount       NUMERIC(18,2),
    margin_amount     NUMERIC(18,2),
    load_batch_id     VARCHAR(50)
);
```

Langkah 5 — Muat CSV ke *Staging*

Pada PostgreSQL, `COPY FROM` dapat memuat data dari file ke tabel. Dokumentasi PostgreSQL menjelaskan bahwa `COPY FROM` menyalin data dari file ke tabel dan menambahkan data ke isi tabel yang sudah ada. (PostgreSQL)

Contoh SQL:

```
COPY stg_sales_raw (
    sales_id,
    line_number,
    transaction_date,
    customer_id,
    product_id,
    branch_code,
    quantity,
    gross_amount,
    discount_amount,
    cost_amount
)
FROM '/data/sales.csv'
WITH (FORMAT csv, HEADER true);
```

Jika menggunakan proses berbasis Python, alurnya dapat dibuat seperti ini.

Langkah 6 — Contoh Python Sederhana untuk Load CSV

```
import csv
import psycopg2
from datetime import datetime

DB_CONFIG = {
    "host": "localhost",
    "dbname": "dw_demo",
    "user": "postgres",
    "password": "postgres",
    "port": 5432,
}

SOURCE_FILE = "sales.csv"
BATCH_ID = datetime.now().strftime("%Y%m%d%H%M%S")
PROCESS_NAME = "load_sales_csv_to_staging"

def main():
    conn = psycopg2.connect(**DB_CONFIG)
    cur = conn.cursor()

    row_count = 0

    try:
        cur.execute("""
            INSERT INTO etl_batch_log
            (batch_id, process_name, source_file, start_time,
status, row_in, row_out, error_count)
            VALUES (%s, %s, %s, CURRENT_TIMESTAMP, %s, 0, 0,
0)
        """, (BATCH_ID, PROCESS_NAME, SOURCE_FILE,
"running"))

        with open(SOURCE_FILE, newline="", encoding="utf-8")
as csvfile:
            reader = csv.DictReader(csvfile)

            for row in reader:
                cur.execute("""
                    INSERT INTO stg_sales_raw (
                        sales_id,
                        line_number,
                        transaction_date,
                        customer_id,
                        product_id,
                        branch_code,
                        quantity,
                        gross_amount,
```

```
                        discount_amount,
                        cost_amount,
                        source_file,
                        load_batch_id
                    )
                    VALUES
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)
                """, (
                    row["sales_id"],
                    int(row["line_number"]),
                    row["transaction_date"],
                    row["customer_id"],
                    row["product_id"],
                    row["branch_code"],
                    float(row["quantity"]),
                    float(row["gross_amount"]),
                    float(row["discount_amount"]),
                    float(row["cost_amount"]),
                    SOURCE_FILE,
                    BATCH_ID
                ))
                row_count += 1

        cur.execute("""
            UPDATE etl_batch_log
            SET end_time = CURRENT_TIMESTAMP,
                status = %s,
                row_in = %s,
                row_out = %s,
                error_count = 0
            WHERE batch_id = %s
        """, ("success", row_count, row_count, BATCH_ID))

        conn.commit()

    except Exception as e:
        conn.rollback()

        cur.execute("""
            INSERT INTO etl_error_log
            (batch_id, process_name, source_key, error_type,
error_message)
            VALUES (%s, %s, %s, %s, %s)
        """, (BATCH_ID, PROCESS_NAME, SOURCE_FILE,
"LOAD_ERROR", str(e)))

        cur.execute("""
            UPDATE etl_batch_log
```

```
                SET end_time = CURRENT_TIMESTAMP,
                    status = %s,
                    row_in = %s,
                    row_out = 0,
                    error_count = 1,
                    notes = %s
                WHERE batch_id = %s
            """, ("failed", row_count, str(e), BATCH_ID))

            conn.commit()
            raise

        finally:
            cur.close()
            conn.close()

if __name__ == "__main__":
    main()
```

Langkah 7 — Transformasi dari *Raw* ke *Clean Staging*

```
INSERT INTO stg_sales_clean (
    sales_id,
    line_number,
    transaction_date,
    customer_id,
    product_id,
    branch_code,
    quantity,
    gross_amount,
    discount_amount,
    net_amount,
    cost_amount,
    margin_amount,
    load_batch_id
)
SELECT
    TRIM(sales_id),
    line_number,
    CAST(transaction_date AS DATE),
    TRIM(customer_id),
    TRIM(product_id),
    UPPER(TRIM(branch_code)),
    quantity,
    gross_amount,
    discount_amount,
    gross_amount - discount_amount AS net_amount,
    cost_amount,
    (gross_amount - discount_amount) - cost_amount AS
margin_amount,
    load_batch_id
FROM stg_sales_raw
WHERE load_batch_id = 'ISI_BATCH_ID'
  AND sales_id IS NOT NULL
  AND transaction_date IS NOT NULL
  AND gross_amount >= 0
  AND quantity > 0;
```

Langkah 8 — Catat Error Data yang Tidak Valid

```
INSERT INTO etl_error_log (
    batch_id,
    process_name,
    source_key,
    error_type,
    error_message
)
SELECT
    load_batch_id,
    'transform_sales_raw_to_clean',
    sales_id,
    'VALIDATION_ERROR',
    'Data tidak valid: field wajib kosong, quantity <= 0,
atau gross_amount negatif'
FROM stg_sales_raw
WHERE load_batch_id = 'ISI_BATCH_ID'
  AND (
      sales_id IS NULL
      OR transaction_date IS NULL
      OR gross_amount < 0
      OR quantity <= 0
  );
```

Langkah 9 — Load ke *Fact Table*

Misalnya sudah ada dimensi:

- `dim_date`
- `dim_customer`
- `dim_product`
- `dim_region`

Maka load ke `fact_sales`:

```
INSERT INTO fact_sales (
    sales_id,
    line_number,
    date_key,
    customer_key,
    product_key,
    region_key,
    quantity,
    gross_amount,
    discount_amount,
    net_amount,
    cost_amount,
    margin_amount
)
SELECT
    s.sales_id,
    s.line_number,
    d.date_key,
    c.customer_key,
    p.product_key,
    r.region_key,
    s.quantity,
    s.gross_amount,
    s.discount_amount,
    s.net_amount,
    s.cost_amount,
    s.margin_amount
FROM stg_sales_clean s
JOIN dim_date d
    ON s.transaction_date = d.full_date
JOIN dim_customer c
    ON s.customer_id = c.customer_id
   AND c.current_flag = 'Y'
JOIN dim_product p
    ON s.product_id = p.product_id
JOIN dim_region r
    ON s.branch_code = r.branch_code
WHERE s.load_batch_id = 'ISI_BATCH_ID';
```

Langkah 10 — Rekonsiliasi

Cek jumlah baris:

```
SELECT COUNT(*) AS raw_count
FROM stg_sales_raw
WHERE load_batch_id = 'ISI_BATCH_ID';

SELECT COUNT(*) AS clean_count
FROM stg_sales_clean
WHERE load_batch_id = 'ISI_BATCH_ID';
```

Cek total nilai:

```
SELECT SUM(gross_amount) AS raw_gross
FROM stg_sales_raw
WHERE load_batch_id = 'ISI_BATCH_ID';

SELECT SUM(gross_amount) AS clean_gross
FROM stg_sales_clean
WHERE load_batch_id = 'ISI_BATCH_ID';
```

Cek data yang gagal *join* ke dimensi:

```
SELECT s.*
FROM stg_sales_clean s
LEFT JOIN dim_product p ON s.product_id = p.product_id
WHERE s.load_batch_id = 'ISI_BATCH_ID'
  AND p.product_key IS NULL;
```

Checklist Desain *Pipeline ETL/ELT*

Area	Pertanyaan
Extract	Apakah sumber data sudah jelas?
Extract	Apakah metode full, incremental, CDC, API, atau file sudah dipilih?
Extract	Apakah ada watermark?
Transform	Apakah aturan cleansing terdokumentasi?
Transform	Apakah standarisasi kode sudah disetujui?
Transform	Apakah deduplikasi punya aturan jelas?
Transform	Apakah data sensitif dimasking?
Load	Apakah load memakai transaksi database?
Load	Apakah ada tabel error?
Load	Apakah ada mekanisme retry dan rollback?
Audit	Apakah row count dicatat?
Audit	Apakah error count dicatat?
Audit	Apakah start/end time dicatat?
Audit	Apakah rekonsiliasi dilakukan?
Metadata	Apakah transformasi terdokumentasi?
Operasi	Apakah ada notifikasi jika gagal?

Kesalahan Umum dalam *ETL/ELT*

Kesalahan yang sering terjadi:

- membuat *pipeline* tanpa log;
- tidak mencatat jumlah baris;
- tidak ada tabel error;
- tidak ada rekonsiliasi;
- transformasi hanya dipahami pembuat kode;
- aturan bisnis tidak disetujui *data owner*;
- semua error langsung diabaikan;
- tidak ada *watermark* untuk incremental load;
- data sensitif masuk mentah tanpa masking;
- file yang sama diproses berkali-kali;
- tidak ada dokumentasi *source-to-target mapping*;
- tidak ada proses ulang yang aman.

Prinsip kritis: *pipeline* yang hanya "jalan" belum tentu profesional. *Pipeline* profesional harus **terukur, terdokumentasi, dapat diaudit, dan aman saat gagal**.

Ringkasan Bab

ETL dan *ELT* adalah jantung operasional *Data Warehouse*. *ETL* cocok ketika data harus dibersihkan dan divalidasi sebelum masuk *warehouse*. *ELT* cocok ketika volume data besar, platform target kuat, dan organisasi membutuhkan fleksibilitas transformasi. Keduanya membutuhkan *governance*, metadata, logging, validasi, dan rekonsiliasi yang kuat.

Poin penting:

- **ETL = transformasi sebelum load; ELT = load dulu, transformasi kemudian.**
- *Extract* dapat dilakukan melalui *full extract*, *incremental extract*, CDC, API, atau file.
- *Transform* mencakup *cleansing*, standardisasi, deduplikasi, *join*, agregasi, aturan bisnis, dan masking.
- *Load* mencakup *initial load*, *incremental load*, *batch load*, *near-real-time load*, dan *error handling*.
- *Logging* wajib mencatat *row count*, *error count*, waktu proses, status, checksum, dan hasil rekonsiliasi.
- Rekonsiliasi adalah bukti bahwa data target sesuai dengan sumber.
- *Pipeline* yang baik harus bisa dijalankan ulang tanpa merusak data.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Jelaskan perbedaan utama antara *ETL* dan *ELT*.
2. Kapan *ETL* lebih tepat digunakan daripada *ELT*?
3. Apa risiko utama jika *ELT* dilakukan tanpa *governance*?
4. Jelaskan perbedaan *full extract* dan *incremental extract*.
5. Apa fungsi *Change Data Capture*?
6. Mengapa *data masking* penting dalam proses transformasi?
7. Apa fungsi *row count* dalam audit *ETL*?
8. Mengapa rekonsiliasi wajib dilakukan pada data transaksi penting?

B. Studi Kasus

Sebuah organisasi ingin memuat data penjualan harian dari file CSV. File dikirim setiap malam oleh cabang. Masalah yang sering terjadi adalah format tanggal tidak konsisten, transaksi ganda, kode cabang salah, dan beberapa nilai transaksi kosong.

Tugas:

- tentukan metode ekstraksi yang sesuai;
- buat aturan *cleansing*;
- buat aturan deduplikasi;
- tentukan tabel *staging*;
- tentukan data yang harus masuk tabel error;
- buat daftar log yang wajib dicatat;
- jelaskan cara rekonsiliasi.

C. Praktik

Buat rancangan *pipeline* sederhana dengan format berikut:

Komponen	Isi
Nama proses	
Sumber data	
Metode extract	full / incremental / CDC / API / file
Tabel staging	
Aturan cleansing	
Aturan transformasi	
Tabel target	
Error handling	
Logging	
Rekonsiliasi	
Frekuensi eksekusi	

BAB 7. Implementasi Data Warehouse: Step-by-Step dari Nol

Implementasi *Data Warehouse* bukan pekerjaan sekali jadi. Ia adalah proses bertahap yang menggabungkan kebutuhan bisnis, desain data, *pipeline*, keamanan, kualitas data, pengujian, dokumentasi, dan operasi harian.

Kesalahan umum dalam proyek *Data Warehouse* adalah langsung membuat tabel dan *dashboard* tanpa fondasi. Hasilnya tampak cepat di awal, tetapi sulit diaudit, lambat dikembangkan, dan rentan menghasilkan angka yang salah.

Referensi *Mastering Data Warehousing* menempatkan pembangunan strategi *Data Warehouse* sebagai proses yang dimulai dari kebutuhan bisnis, pemilihan alat dan teknologi, lalu pembuatan *roadmap* yang realistis. Referensi lain menekankan bahwa implementasi *Data Warehouse* perlu mencakup sumber data, *ETL*, *staging*, database, metadata, reporting, access control, kualitas data, performance enhancement, maintenance, dan change management.

Golden Rule: implementasi *Data Warehouse* yang baik harus **kecil dulu, benar dulu, terdokumentasi dulu, lalu diperluas**.

7.1 Tahapan Implementasi

Implementasi *Data Warehouse* profesional dapat dibagi menjadi 12 tahap:

```
Project Charter
    ↓
Requirement Gathering
    ↓
Data Profiling
    ↓
Logical Design
    ↓
Physical Design
    ↓
ETL Development
    ↓
Data Quality Testing
    ↓
Security Setup
    ↓
UAT
    ↓
Deployment
    ↓
Monitoring
    ↓
Maintenance
```

Referensi implementasi yang digunakan menekankan pentingnya tujuan yang jelas, analisis data, pemodelan, integrasi *ETL*, arsitektur, metadata, keamanan, pengujian, rekonsiliasi, pelatihan, dokumentasi, monitoring, backup, maintenance, dan perbaikan berkelanjutan.

A. *Project Charter*

Project charter adalah dokumen awal yang menyatakan mengapa proyek *Data Warehouse* dibuat, siapa sponsornya, apa ruang lingkupnya, dan bagaimana keberhasilannya diukur.

Isi minimal:

Elemen	Isi
Nama proyek	Data Warehouse Penjualan
Sponsor	Direksi / Kepala Unit
Masalah bisnis	laporan lambat dan tidak konsisten
Tujuan	satu sumber data resmi untuk analitik penjualan
Scope awal	penjualan, pelanggan, produk, cabang
Output	*FactSales*, dimensi utama, dashboard penjualan
Timeline	MVP 4–6 minggu
Risiko	kualitas data, duplikasi, perubahan format file
Kriteria sukses	dashboard disetujui pengguna dan angka tervalidasi

Prinsip kritis: tanpa *project charter*, proyek mudah melebar dan berubah menjadi kumpulan permintaan laporan tanpa arah.

B. *Requirement Gathering*

Requirement gathering menerjemahkan kebutuhan bisnis menjadi kebutuhan data dan teknis.

Pertanyaan utama:

- laporan apa yang dibutuhkan?
- keputusan apa yang ingin didukung?
- siapa pengguna utama?
- apa KPI utama?
- sumber data apa yang tersedia?
- seberapa sering data diperbarui?
- data mana yang sensitif?
- berapa lama data disimpan?
- bagaimana SLA dashboard?

Output:

- *business requirement*;
- *data requirement*;
- *security requirement*;
- *refresh requirement*;
- *retention requirement*;
- daftar KPI;
- daftar sumber data.

C. *Data Profiling*

Data profiling adalah proses memahami kondisi data sumber sebelum desain dan *ETL* dibuat. Tahap ini sering diabaikan, padahal sangat penting.

Yang diperiksa:

- jumlah baris;
- kolom kosong;
- duplikasi;
- tipe data;
- nilai ekstrim;
- format tanggal;
- kode tidak valid;
- data historis;
- konsistensi antar sumber.

Contoh pemeriksaan:

```
SELECT COUNT(*) AS total_rows FROM stg_sales_raw;

SELECT COUNT(*) AS null_customer
FROM stg_sales_raw
WHERE customer_id IS NULL;

SELECT sales_id, line_number, COUNT(*)
FROM stg_sales_raw
GROUP BY sales_id, line_number
HAVING COUNT(*) > 1;
```

Referensi desain *Data Warehouse* menekankan pentingnya identifikasi sumber data dan penilaian kualitas data, termasuk missing values, duplikasi, inkonsistensi, dan kebutuhan *cleansing*.

Prinsip kritis: jangan mendesain *Data Warehouse* berdasarkan asumsi data bersih. Selalu profil data terlebih dahulu.

D. *Logical Design*

Logical design adalah rancangan konseptual dan logis dari model data. Pada tahap ini, fokusnya adalah hubungan bisnis antar data, bukan detail tipe data fisik.

Output:

- proses bisnis yang dimodelkan;
- *grain*;
- *fact table*;
- *dimension table*;
- hierarki;
- KPI dan *measure*;
- relasi antar entitas.

Contoh:

```
FactSales
Grain: 1 baris = 1 item transaksi penjualan

Dimensi:
- DimCustomer
- DimProduct
- DimDate
- DimBranch

Measure:
- quantity
- gross_amount
- discount_amount
- net_amount
- margin_amount
```

Referensi yang digunakan menjelaskan bahwa *logical design* menerjemahkan kebutuhan bisnis menjadi model konseptual yang menunjukkan entitas, atribut, dan relasi untuk kebutuhan analisis dan pelaporan.

E. *Physical Design*

Physical design menerjemahkan model logis menjadi struktur database nyata.

Output:

- nama tabel;
- nama kolom;
- tipe data;
- *primary key*;
- *foreign key*;
- indeks;
- partisi;
- skema database;
- tabel *staging*;
- tabel error;
- tabel log;
- struktur backup.

Referensi yang digunakan menjelaskan bahwa *physical data model* memetakan model logis ke struktur fisik database, termasuk tabel, kolom, tipe data, constraint, primary key, foreign key, dan relasi.

Contoh keputusan fisik:

Area	Keputusan
Skema	`raw`, `staging`, `dw`, `mart`, `audit`
Tipe data uang	`NUMERIC(18,2)`
Tipe data tanggal	`DATE`
Kunci dimensi	integer *surrogate key*
Partisi fakta	berdasarkan bulan transaksi
Indeks	kolom tanggal, produk, cabang

F. *ETL Development*

ETL development membangun proses pengambilan, transformasi, dan pemuatan data.

Output:

- script extract;
- script transform;
- script load;
- *source-to-target mapping*;
- log proses;
- tabel error;
- validasi;
- rekonsiliasi.

Alur dasar:

```
CSV / Database / API
        ↓
Raw Staging
        ↓
Clean Staging
        ↓
Dimension Load
        ↓
Fact Load
        ↓
Data Mart
```

PostgreSQL menyediakan perintah `COPY FROM` untuk memuat data dari file ke tabel, dan `COPY TO` untuk menyalin isi tabel atau hasil query ke file. (PostgreSQL)

G. *Data Quality Testing*

Pengujian kualitas data memastikan data yang masuk benar, lengkap, konsisten, dan siap dipakai.

Pemeriksaan minimal:

Dimensi Kualitas	Contoh Tes
Completeness	kolom wajib tidak boleh kosong
Uniqueness	`sales_id + line_number` harus unik
Validity	tanggal valid, amount tidak negatif
Consistency	kode cabang ada di referensi
Accuracy	total penjualan cocok dengan sumber
Timeliness	data harian masuk sebelum jam tertentu

Konsep *data tests* dalam praktik modern biasanya berupa query yang mencari baris gagal. Jika query menghasilkan nol baris gagal, tes dianggap lolos. Dokumentasi dbt menjelaskan bahwa tes data dapat memeriksa nilai unik, tidak kosong, relasi antar model, dan nilai yang diterima sesuai daftar. (dbt Developer Hub)

H. *Security Setup*

Keamanan harus disiapkan sebelum sistem dipakai pengguna.

Yang perlu dibuat:

- role pengguna;
- matriks akses;
- pembatasan data sensitif;
- audit log;
- aturan ekspor;
- masking;
- backup terenkripsi jika dibutuhkan;
- pemisahan akses *developer*, analis, auditor, dan pengguna bisnis.

Contoh matriks akses:

Role	Raw Data	Staging	DW Core	Data Mart	Dashboard
Data Engineer	Ya	Ya	Ya	Ya	Terbatas
Analis	Tidak	Tidak	Terbatas	Ya	Ya
Direksi	Tidak	Tidak	Tidak	Ringkasan	Ya
Auditor	Terbatas	Terbatas	Ya	Ya	Ya
Admin	Ya	Ya	Ya	Ya	Ya

Referensi arsitektur *cloud data warehouse* menekankan perlunya kebijakan akses berbasis peran, masking, audit, monitoring, dan kepatuhan untuk melindungi data sensitif tanpa mengorbankan kegunaan.

I. *User Acceptance Testing* / UAT

UAT adalah tahap pengguna bisnis memvalidasi apakah sistem sudah sesuai kebutuhan.

Yang diuji:

- angka KPI;
- filter dashboard;
- definisi laporan;
- akses pengguna;
- performa;
- kesesuaian periode;
- kesesuaian sumber data;
- rekonsiliasi dengan laporan lama;
- kemudahan penggunaan.

Contoh UAT:

Skenario	Ekspektasi	Status
Direksi melihat penjualan bulan ini	angka sama dengan laporan resmi	
Manajer cabang memfilter wilayah	hanya data wilayahnya tampil	
Analis melihat margin produk	rumus margin benar	
Auditor cek asal data	lineage tersedia	

Prinsip kritis: UAT bukan demo. UAT adalah **pembuktian bahwa sistem layak menjadi rujukan keputusan**.

J. *Deployment*

Deployment memindahkan sistem dari lingkungan pengembangan ke produksi.

Checklist deployment:

- struktur database siap;
- data historis sudah dimuat;
- ETL terjadwal;
- kualitas data lolos;
- role akses aktif;
- dashboard tersedia;
- backup berjalan;
- monitoring aktif;
- dokumentasi selesai;
- UAT disetujui;
- rollback plan tersedia.

Strategi deployment:

Strategi	Cocok Untuk
Big bang	sistem kecil, risiko rendah
Parallel run	membandingkan laporan lama dan baru
Phased rollout	banyak unit / cabang
Pilot production	pengguna terbatas sebelum perluasan

K. Monitoring

Monitoring memastikan *Data Warehouse* tetap sehat setelah produksi.

Yang dipantau:

- status ETL;
- durasi ETL;
- jumlah baris;
- error count;
- storage;
- performa query;
- akses pengguna;
- gagal login;
- dashboard lambat;
- data terlambat;
- backup berhasil/gagal.

Referensi implementasi menekankan monitoring, backup, maintenance, performance tuning, rekonsiliasi, audit, feedback pengguna, dan perbaikan berkelanjutan sebagai bagian dari siklus implementasi yang dinamis.

L. Maintenance

Maintenance menjaga sistem tetap relevan dan stabil.

Kegiatan:

- update dimensi;
- perbaikan ETL;
- optimasi query;
- arsip data lama;
- pembersihan staging;
- pembaruan metadata;
- review akses;
- backup restore test;
- penyesuaian KPI;
- penambahan data mart;
- dokumentasi perubahan.

Referensi juga menekankan bahwa maintenance dan change management mencakup update, version control, jadwal refresh, backup and recovery, dan tugas pemeliharaan agar *Data Warehouse* tetap berjalan efisien.

7.2 Contoh Proyek Mini: *Data Warehouse* Penjualan

Kasus mini ini digunakan untuk praktik sertifikasi. Tujuannya adalah membangun *Data Warehouse* sederhana dari file CSV sampai dashboard penjualan.

A. Tabel Sumber

File sumber:

```
customer.csv
product.csv
sales.csv
branch.csv
```

Isi minimal:

customer.csv

customer_id	customer_name	customer_type	city	province
C001	PT Nusantara Maju	Corporate	Jakarta	DKI Jakarta
C002	CV Sinar Abadi	Retail	Bandung	Jawa Barat

product.csv

product_id	product_name	category	unit	cost_price
P001	Produk A	Material	unit	700000
P002	Produk B	Peralatan	unit	300000

branch.csv

branch_code	branch_name	city	province	region_name
JKT01	Cabang Jakarta 1	Jakarta	DKI Jakarta	Barat
BDG01	Cabang Bandung 1	Bandung	Jawa Barat	Barat

sales.csv

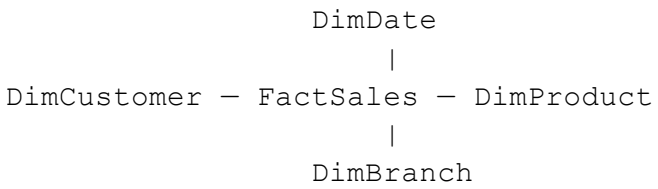
sales_id	line_number	sales_date	customer_id	product_id	branch_code	quantity	gross_amount	discount_amount
S001	1	2026-01-01	C001	P001	JKT01	2	2000000	100000
S001	2	2026-01-01	C001	P002	JKT01	1	500000	0
S002	1	2026-01-02	C002	P001	BDG01	1	1000000	50000

B. Output *Data Warehouse*

Tabel target:

```
DimCustomer
DimProduct
DimDate
DimBranch
FactSales
```

Model:



C. KPI Dashboard

Dashboard penjualan minimal berisi:

- total penjualan bersih;
- total margin;
- penjualan per bulan;
- penjualan per cabang;
- penjualan per produk;
- top pelanggan;
- diskon total;
- margin persen.

Rumus:

```
net_amount = gross_amount - discount_amount
margin_amount = net_amount - cost_amount
margin_percent = margin_amount / net_amount × 100
```

7.3 Struktur Folder Proyek

Struktur folder membantu proyek tetap rapi, mudah diaudit, dan mudah dipelihara.

```
datawarehouse_project/
├── data/
│   ├── raw/
│   │   ├── customer.csv
│   │   ├── product.csv
│   │   ├── sales.csv
│   │   └── branch.csv
│   └── staging/
├── sql/
│   ├── 01_create_schema.sql
│   ├── 02_create_staging_tables.sql
│   ├── 03_create_dimension_tables.sql
│   ├── 04_create_fact_tables.sql
│   ├── 05_load_dimensions.sql
│   ├── 06_load_facts.sql
│   └── 07_create_views.sql
├── etl/
│   ├── load_csv_to_staging.py
│   ├── transform_sales.py
│   └── run_etl.sh
├── docs/
│   ├── data_dictionary.md
│   ├── etl_mapping.md
│   ├── data_lineage.md
│   ├── access_matrix.md
│   └── operational_manual.md
├── tests/
│   ├── test_row_count.sql
│   ├── test_null_values.sql
│   ├── test_duplicate_sales.sql
│   └── test_reconciliation.sql
├── dashboards/
│   └── dashboard_spec.md
└── logs/
    ├── etl_batch.log
    └── error.log
```

Fungsi Folder

Folder	Fungsi
`data/raw/`	menyimpan file mentah
`data/staging/`	hasil sementara atau file validasi
`sql/`	semua definisi tabel, view, load
`etl/`	script pemrosesan
`docs/`	dokumentasi proyek
`tests/`	query pengujian
`dashboards/`	spesifikasi dashboard
`logs/`	catatan proses dan error

Prinsip kritis: proyek *Data Warehouse* profesional harus dapat dipahami oleh orang lain, bukan hanya oleh pembuat awal.

7.4 Standar Dokumentasi

Dokumentasi bukan pekerjaan tambahan. Dokumentasi adalah bagian dari sistem. Tanpa dokumentasi, *Data Warehouse* sulit diaudit, sulit dirawat, dan sulit dikembangkan.

A. *Data Dictionary*

Data dictionary menjelaskan arti tabel dan kolom.

Contoh:

Tabel	Kolom	Definisi	Tipe	Aturan
FactSales	net_amount	penjualan setelah diskon	numeric	>= 0
FactSales	margin_amount	laba kotor transaksi	numeric	net - cost
DimCustomer	customer_id	kode pelanggan sumber	varchar	wajib
DimProduct	product_id	kode produk sumber	varchar	wajib
DimBranch	branch_code	kode cabang	varchar	wajib valid

B. *ETL Mapping*

ETL mapping menjelaskan hubungan dari sumber ke target.

Source	Kolom Source	Target	Kolom Target	Transformasi
sales.csv	sales_date	FactSales	date_key	lookup ke DimDate
sales.csv	customer_id	FactSales	customer_key	lookup ke DimCustomer
sales.csv	product_id	FactSales	product_key	lookup ke DimProduct
sales.csv	gross_amount	FactSales	gross_amount	langsung
sales.csv	discount_amount	FactSales	discount_amount	langsung
sales.csv	gross - discount	FactSales	net_amount	hitung

C. *Data Lineage*

Data lineage menunjukkan asal dan perjalanan data.

Contoh:

```
sales.csv
    ↓
stg_sales_raw
    ↓
stg_sales_clean
    ↓
FactSales
    ↓
mart_sales_summary
    ↓
Dashboard Penjualan
```

Referensi metadata menekankan pentingnya data source, transformasi, business rule, dan *lineage* agar pengguna memahami dan menafsirkan data dalam *warehouse*.

D. *Access Matrix*

Access matrix menjelaskan siapa boleh melihat apa.

Role	FactSales	DimCustomer	DimProduct	Dashboard	Export
Direksi	Ringkasan	Masked	Ya	Ya	Tidak
Manajer Cabang	Cabangnya saja	Masked	Ya	Ya	Terbatas
Analis	Ya	Masked	Ya	Ya	Ya
Data Engineer	Ya	Ya	Ya	Terbatas	Tidak
Auditor	Ya	Ya	Ya	Ya	Terbatas

E. *Test Case*

Test case membuktikan bahwa data dan fungsi sistem benar.

Test Case	Tujuan	Ekspektasi
TC-001	cek transaksi duplikat	tidak ada duplikasi
TC-002	cek nilai kosong	kolom wajib tidak kosong
TC-003	cek total penjualan	sama dengan sumber
TC-004	cek akses user	role sesuai hak akses
TC-005	cek dashboard	KPI tampil benar

F. *Deployment Checklist*

Checklist sebelum produksi:

Item	Status
Project charter disetujui	
Requirement disetujui	
Model data selesai	
Tabel dibuat	
ETL berjalan	
Data quality test lolos	
Rekonsiliasi selesai	
Access matrix diterapkan	
Dashboard diuji	
UAT disetujui	
Backup aktif	
Monitoring aktif	
Dokumentasi lengkap	
Rollback plan tersedia	

G. *Operational Manual*

Operational manual menjelaskan cara mengoperasikan sistem.

Isi minimal:

- cara menjalankan ETL;
- jadwal ETL;
- lokasi file input;
- cara membaca log;
- cara menangani error;
- cara melakukan rekonsiliasi;
- cara menambah user;
- cara restore backup;
- kontak penanggung jawab;
- prosedur eskalasi.

Hands-on. Membangun Mini *Data Warehouse* dari CSV sampai Dashboard

Bagian ini memberi alur praktik lengkap. Fokusnya adalah *workflow* yang dapat dipakai untuk lab sertifikasi.

Langkah 1 — Buat Database dan Skema

Gunakan PostgreSQL atau MariaDB. Contoh berikut memakai SQL standar yang mudah disesuaikan.

```
CREATE SCHEMA IF NOT EXISTS raw;
CREATE SCHEMA IF NOT EXISTS staging;
CREATE SCHEMA IF NOT EXISTS dw;
CREATE SCHEMA IF NOT EXISTS mart;
CREATE SCHEMA IF NOT EXISTS audit;
```

Langkah 2 — Buat Tabel *Raw Staging*

```
CREATE TABLE raw.customer_raw (
    customer_id     VARCHAR(50),
    customer_name   VARCHAR(200),
    customer_type   VARCHAR(50),
    city            VARCHAR(100),
    province        VARCHAR(100),
    source_file     VARCHAR(200),
    loaded_at       TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE raw.product_raw (
    product_id      VARCHAR(50),
    product_name    VARCHAR(200),
    category        VARCHAR(100),
    unit_name       VARCHAR(50),
    cost_price      NUMERIC(18,2),
    source_file     VARCHAR(200),
    loaded_at       TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE TABLE raw.branch_raw (
    branch_code     VARCHAR(50),
    branch_name     VARCHAR(200),
    city            VARCHAR(100),
    province        VARCHAR(100),
    region_name     VARCHAR(100),
    source_file     VARCHAR(200),
    loaded_at       TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE raw.sales_raw (
    sales_id          VARCHAR(50),
    line_number       INTEGER,
    sales_date        VARCHAR(50),
    customer_id       VARCHAR(50),
    product_id        VARCHAR(50),
    branch_code       VARCHAR(50),
    quantity          NUMERIC(18,2),
    gross_amount      NUMERIC(18,2),
    discount_amount   NUMERIC(18,2),
    source_file       VARCHAR(200),
    loaded_at         TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Langkah 3 — Load CSV ke *Raw Staging*

Contoh menggunakan `COPY FROM` pada PostgreSQL:

```
COPY raw.customer_raw(customer_id, customer_name,
customer_type, city, province)
FROM '/data/raw/customer.csv'
WITH (FORMAT csv, HEADER true);

COPY raw.product_raw(product_id, product_name, category,
unit_name, cost_price)
FROM '/data/raw/product.csv'
WITH (FORMAT csv, HEADER true);

COPY raw.branch_raw(branch_code, branch_name, city, province,
region_name)
FROM '/data/raw/branch.csv'
WITH (FORMAT csv, HEADER true);

COPY raw.sales_raw(sales_id, line_number, sales_date,
customer_id, product_id, branch_code, quantity, gross_amount,
discount_amount)
FROM '/data/raw/sales.csv'
WITH (FORMAT csv, HEADER true);
```

PostgreSQL menjelaskan bahwa `COPY` memindahkan data antara tabel dan file; `COPY FROM` memuat data dari file ke tabel. (PostgreSQL)

Langkah 4 — Buat Tabel Dimensi

```
CREATE TABLE dw.dim_customer (
    customer_key    SERIAL PRIMARY KEY,
    customer_id     VARCHAR(50) NOT NULL,
    customer_name   VARCHAR(200),
    customer_type   VARCHAR(50),
    city            VARCHAR(100),
    province        VARCHAR(100),
    current_flag    CHAR(1) DEFAULT 'Y'
);

CREATE TABLE dw.dim_product (
    product_key     SERIAL PRIMARY KEY,
    product_id      VARCHAR(50) NOT NULL,
    product_name    VARCHAR(200),
    category        VARCHAR(100),
    unit_name       VARCHAR(50),
    cost_price      NUMERIC(18,2)
);

CREATE TABLE dw.dim_branch (
    branch_key      SERIAL PRIMARY KEY,
    branch_code     VARCHAR(50) NOT NULL,
    branch_name     VARCHAR(200),
    city            VARCHAR(100),
    province        VARCHAR(100),
    region_name     VARCHAR(100)
);

CREATE TABLE dw.dim_date (
    date_key        INTEGER PRIMARY KEY,
    full_date       DATE NOT NULL,
    day_number      INTEGER,
    month_number    INTEGER,
    month_name      VARCHAR(20),
    quarter_number  INTEGER,
    year_number     INTEGER
);
```

Langkah 5 — Buat Tabel Fakta

```
CREATE TABLE dw.fact_sales (
    sales_fact_key    BIGSERIAL PRIMARY KEY,
    sales_id          VARCHAR(50) NOT NULL,
    line_number       INTEGER NOT NULL,
    date_key          INTEGER NOT NULL,
    customer_key      INTEGER NOT NULL,
    product_key       INTEGER NOT NULL,
    branch_key        INTEGER NOT NULL,
    quantity          NUMERIC(18,2),
    gross_amount      NUMERIC(18,2),
    discount_amount   NUMERIC(18,2),
    net_amount        NUMERIC(18,2),
    cost_amount       NUMERIC(18,2),
    margin_amount     NUMERIC(18,2),

    FOREIGN KEY (date_key) REFERENCES dw.dim_date(date_key),
    FOREIGN KEY (customer_key) REFERENCES
dw.dim_customer(customer_key),
    FOREIGN KEY (product_key) REFERENCES
dw.dim_product(product_key),
    FOREIGN KEY (branch_key) REFERENCES
dw.dim_branch(branch_key)
);
```

Langkah 6 — Load Dimensi

Load *DimCustomer*

```
INSERT INTO dw.dim_customer (
    customer_id,
    customer_name,
    customer_type,
    city,
    province
)
SELECT DISTINCT
    TRIM(customer_id),
    TRIM(customer_name),
    TRIM(customer_type),
    TRIM(city),
    TRIM(province)
FROM raw.customer_raw
WHERE customer_id IS NOT NULL;
```

Load *DimProduct*

```
INSERT INTO dw.dim_product (
    product_id,
    product_name,
    category,
    unit_name,
    cost_price
)
SELECT DISTINCT
    TRIM(product_id),
    TRIM(product_name),
    TRIM(category),
    TRIM(unit_name),
    cost_price
FROM raw.product_raw
WHERE product_id IS NOT NULL;
```

Load *DimBranch*

```
INSERT INTO dw.dim_branch (
    branch_code,
    branch_name,
    city,
    province,
    region_name
)
SELECT DISTINCT
    UPPER(TRIM(branch_code)),
    TRIM(branch_name),
    TRIM(city),
    TRIM(province),
    TRIM(region_name)
FROM raw.branch_raw
WHERE branch_code IS NOT NULL;
```

Load *DimDate* dari Data Penjualan

```
INSERT INTO dw.dim_date (
    date_key,
    full_date,
    day_number,
    month_number,
    month_name,
    quarter_number,
    year_number
)
SELECT DISTINCT
    TO_CHAR(CAST(sales_date AS DATE), 'YYYYMMDD')::INTEGER AS
date_key,
    CAST(sales_date AS DATE) AS full_date,
    EXTRACT(DAY FROM CAST(sales_date AS DATE))::INTEGER AS
day_number,
    EXTRACT(MONTH FROM CAST(sales_date AS DATE))::INTEGER AS
month_number,
    TO_CHAR(CAST(sales_date AS DATE), 'Month') AS month_name,
    EXTRACT(QUARTER FROM CAST(sales_date AS DATE))::INTEGER
AS quarter_number,
    EXTRACT(YEAR FROM CAST(sales_date AS DATE))::INTEGER AS
year_number
FROM raw.sales_raw
WHERE sales_date IS NOT NULL;
```

Langkah 7 — Load *FactSales*

```
INSERT INTO dw.fact_sales (
    sales_id,
    line_number,
    date_key,
    customer_key,
    product_key,
    branch_key,
    quantity,
    gross_amount,
    discount_amount,
    net_amount,
    cost_amount,
    margin_amount
)
SELECT
    s.sales_id,
    s.line_number,
    d.date_key,
    c.customer_key,
    p.product_key,
    b.branch_key,
    s.quantity,
    s.gross_amount,
    s.discount_amount,
    s.gross_amount - s.discount_amount AS net_amount,
    p.cost_price * s.quantity AS cost_amount,
    (s.gross_amount - s.discount_amount) - (p.cost_price *
s.quantity) AS margin_amount
FROM raw.sales_raw s
JOIN dw.dim_date d
    ON CAST(s.sales_date AS DATE) = d.full_date
JOIN dw.dim_customer c
    ON TRIM(s.customer_id) = c.customer_id
JOIN dw.dim_product p
    ON TRIM(s.product_id) = p.product_id
JOIN dw.dim_branch b
    ON UPPER(TRIM(s.branch_code)) = b.branch_code
WHERE s.sales_id IS NOT NULL
  AND s.quantity > 0
  AND s.gross_amount >= 0;
```

Langkah 8 — Buat *Data Mart* Penjualan

```
CREATE VIEW mart.sales_summary_monthly AS
SELECT
    d.year_number,
    d.month_number,
    b.region_name,
    b.branch_name,
    p.category,
    SUM(f.quantity) AS total_quantity,
    SUM(f.net_amount) AS total_sales,
    SUM(f.margin_amount) AS total_margin,
    ROUND(
        SUM(f.margin_amount) / NULLIF(SUM(f.net_amount), 0) *
100,
        2
    ) AS margin_percent
FROM dw.fact_sales f
JOIN dw.dim_date d ON f.date_key = d.date_key
JOIN dw.dim_branch b ON f.branch_key = b.branch_key
JOIN dw.dim_product p ON f.product_key = p.product_key
GROUP BY
    d.year_number,
    d.month_number,
    b.region_name,
    b.branch_name,
    p.category;
```

Langkah 9 — Buat Query Dashboard

Total Penjualan

```
SELECT
    SUM(net_amount) AS total_sales,
    SUM(margin_amount) AS total_margin,
    ROUND(SUM(margin_amount) / NULLIF(SUM(net_amount), 0) *
100, 2) AS margin_percent
FROM dw.fact_sales;
```

Penjualan per Bulan

```
SELECT
    d.year_number,
    d.month_number,
    SUM(f.net_amount) AS total_sales
FROM dw.fact_sales f
JOIN dw.dim_date d ON f.date_key = d.date_key
GROUP BY d.year_number, d.month_number
ORDER BY d.year_number, d.month_number;
```

Penjualan per Cabang

```
SELECT
    b.region_name,
    b.branch_name,
    SUM(f.net_amount) AS total_sales,
    SUM(f.margin_amount) AS total_margin
FROM dw.fact_sales f
JOIN dw.dim_branch b ON f.branch_key = b.branch_key
GROUP BY b.region_name, b.branch_name
ORDER BY total_sales DESC;
```

Penjualan per Produk

```
SELECT
    p.category,
    p.product_name,
    SUM(f.quantity) AS total_quantity,
    SUM(f.net_amount) AS total_sales
FROM dw.fact_sales f
JOIN dw.dim_product p ON f.product_key = p.product_key
GROUP BY p.category, p.product_name
ORDER BY total_sales DESC;
```

Langkah 10 — Visualisasi di *Dashboard*

Gunakan *Apache Superset* atau *Metabase*.

Opsi A — *Apache Superset*

Alur:

1. koneksikan Superset ke database;
2. tambahkan tabel atau *view* `mart.sales_summary_monthly`;
3. buat chart penjualan bulanan;
4. buat chart penjualan per cabang;
5. buat chart margin per kategori;
6. gabungkan chart ke dashboard.

Apache Superset mendukung eksplorasi data melalui SQL query, visual query builder, dan visualisasi interaktif. Dokumentasi resminya juga menjelaskan alur membuat dashboard: koneksi ke database, konfigurasi tabel, eksplorasi data, lalu menambahkan visualisasi ke dashboard. (Apache Superset)

Opsi B — *Metabase*

Alur:

1. koneksikan Metabase ke database;
2. buat *question* dari tabel atau SQL;
3. pilih visualisasi;
4. simpan *question*;
5. gabungkan beberapa *question* menjadi dashboard;
6. atur akses koleksi.

Dokumentasi Metabase menjelaskan bahwa *questions* adalah query beserta hasil dan visualisasinya, sedangkan dashboard mengelompokkan beberapa *questions* dalam satu halaman sebagai laporan yang dapat dibagikan. (Metabase)

Langkah 11 — Pengujian Data

Cek Duplikasi Transaksi

```
SELECT sales_id, line_number, COUNT(*)
FROM dw.fact_sales
GROUP BY sales_id, line_number
HAVING COUNT(*) > 1;
```

Ekspektasi: **0 baris**.

Cek Nilai Kosong

```
SELECT COUNT(*) AS missing_dimension
FROM dw.fact_sales
WHERE date_key IS NULL
   OR customer_key IS NULL
   OR product_key IS NULL
   OR branch_key IS NULL;
```

Ekspektasi: **0**.

Rekonsiliasi Total Penjualan

```
SELECT SUM(gross_amount - discount_amount) AS source_total
FROM raw.sales_raw;

SELECT SUM(net_amount) AS warehouse_total
FROM dw.fact_sales;
```

Ekspektasi: **sama**.

Cek Margin Negatif

```
SELECT *
FROM dw.fact_sales
WHERE margin_amount < 0;
```

Ekspektasi: bisa 0 atau ada data yang perlu dijelaskan. Margin negatif belum tentu salah, tetapi harus dapat dijelaskan.

Langkah 12 — Dokumentasi Minimum

Setelah mini *Data Warehouse* berjalan, buat dokumentasi:

Dokumen	Isi
`data_dictionary.md`	definisi tabel dan kolom
`etl_mapping.md`	mapping source ke target
`data_lineage.md`	alur data dari CSV sampai dashboard
`access_matrix.md`	role dan hak akses
`test_case.md`	skenario pengujian
`deployment_checklist.md`	checklist produksi
`operational_manual.md`	cara menjalankan dan memperbaiki proses

Checklist Implementasi Bab 7

Area	Pertanyaan
Project charter	apakah tujuan, sponsor, scope, dan kriteria sukses jelas?
Requirement	apakah KPI, pengguna, sumber data, dan refresh sudah jelas?
Profiling	apakah data kosong, duplikasi, dan format sudah diperiksa?
Logical design	apakah *grain*, *fact*, dan dimensi sudah jelas?
Physical design	apakah tabel, tipe data, key, dan indeks sudah dibuat?
ETL	apakah extract, transform, dan load terdokumentasi?
Quality	apakah ada test null, duplicate, validitas, dan rekonsiliasi?
Security	apakah akses berbasis role sudah diterapkan?
UAT	apakah pengguna bisnis sudah menyetujui angka?
Deployment	apakah backup, monitoring, dan rollback tersedia?
Maintenance	apakah ada prosedur perubahan dan perbaikan?
Dokumentasi	apakah data dictionary, lineage, dan manual tersedia?

Kesalahan Umum Implementasi

Kesalahan yang sering terjadi:

- memulai dari *dashboard*, bukan kebutuhan bisnis;
- tidak membuat *project charter*;
- *scope* terlalu luas;
- tidak melakukan *data profiling*;
- *grain fact table* tidak jelas;
- tidak ada *staging area*;
- tidak ada tabel error dan log;
- tidak melakukan rekonsiliasi;
- akses pengguna tidak dikontrol;
- UAT hanya formalitas;
- deployment tanpa rollback plan;
- dokumentasi tidak dibuat;
- maintenance tidak direncanakan.

Prinsip kritis: implementasi *Data Warehouse* bukan hanya membuat data tampil, tetapi membuat data **benar, terlacak, aman, dan dipercaya**.

Ringkasan Bab

Implementasi *Data Warehouse* harus dilakukan bertahap: *project charter*, requirement, profiling, logical design, physical design, ETL, quality testing, security, UAT, deployment, monitoring, dan maintenance. Proyek mini penjualan menunjukkan bagaimana file CSV dapat diubah menjadi *Data Warehouse* sederhana dengan *FactSales*, dimensi utama, *data mart*, dan dashboard.

Poin penting:

- **Mulai dari scope kecil yang bernilai.**
- *Project charter* mencegah proyek melebar.
- *Data profiling* mencegah desain berbasis asumsi.
- *Logical design* menentukan *grain*, fakta, dan dimensi.
- *Physical design* menerjemahkan model ke tabel nyata.
- *ETL* harus memiliki log, error handling, dan rekonsiliasi.
- UAT harus memvalidasi angka, bukan hanya tampilan.
- Deployment membutuhkan checklist, backup, monitoring, dan rollback.
- Dokumentasi adalah bagian dari sistem, bukan lampiran belakangan.
- Maintenance memastikan *Data Warehouse* tetap relevan dan stabil.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Mengapa *project charter* penting dalam implementasi *Data Warehouse*?
2. Apa perbedaan *logical design* dan *physical design*?
3. Mengapa *data profiling* harus dilakukan sebelum ETL?
4. Apa fungsi *staging area* dalam proyek mini?
5. Mengapa UAT harus dilakukan oleh pengguna bisnis?
6. Apa risiko deployment tanpa rollback plan?
7. Sebutkan minimal lima dokumen wajib dalam proyek *Data Warehouse*.
8. Mengapa rekonsiliasi data penting sebelum dashboard dipakai direksi?

B. Studi Kasus

Sebuah perusahaan ingin membuat mini *Data Warehouse* penjualan dari empat file: `customer.csv`, `product.csv`, `sales.csv`, dan `branch.csv`.

Tugas:

- buat *project charter* singkat;
- tentukan *grain FactSales*;
- buat daftar dimensi;
- buat 5 KPI dashboard;
- tentukan 5 tes kualitas data;
- buat struktur folder proyek;
- buat checklist deployment.

C. Praktik

Buat mini *Data Warehouse* dengan output berikut:

Output	Keterangan
`DimCustomer`	data pelanggan
`DimProduct`	data produk
`DimDate`	dimensi waktu
`DimBranch`	data cabang
`FactSales`	transaksi penjualan
`mart_sales_summary_monthly`	ringkasan penjualan bulanan
Dashboard	penjualan, margin, produk, cabang

Target akhir praktik: peserta mampu menjelaskan **alur data dari CSV sampai dashboard**, lengkap dengan *data dictionary*, *ETL mapping*, *data lineage*, pengujian kualitas data, dan checklist deployment.

BAGIAN III — DEPLOYMENT, OPERASI, SECURITY, GOVERNANCE, DAN OPTIMASI

BAB 8. Deployment Data Warehouse: Dari Development ke Production

Membangun *Data Warehouse* di lingkungan lab relatif mudah. Tantangan sebenarnya dimulai ketika sistem harus masuk ke *production*. Pada tahap ini, data tidak lagi sekadar bahan latihan. Data sudah menjadi dasar laporan manajemen, audit, KPI, analitik bisnis, dan keputusan strategis.

Deployment *Data Warehouse* bukan sekadar menyalin tabel dari satu server ke server lain. Deployment mencakup kesiapan skema, kualitas data, *ETL/ELT*, hak akses, backup, monitoring, dokumentasi, UAT, migrasi awal, strategi *cutover*, dan *rollback plan*.

Referensi *Mastering Data Warehousing* menekankan bahwa fase deployment harus direncanakan, dijadwalkan, diluncurkan ke komunitas pengguna, di monitor, diberi dukungan, lalu dilanjutkan dengan *post-implementation review* dan maintenance. Referensi lain menekankan bahwa implementasi *Data Warehouse* harus mencakup *staging*, *ETL*, database, metadata, reporting, access control, data quality, performance enhancement, maintenance, monitoring, dan keamanan.

Golden Rule: *Deployment* yang baik bukan hanya membuat sistem “naik”, tetapi memastikan sistem **bisa dipercaya, bisa dipulihkan, bisa dimonitor, dan bisa dipertanggungjawabkan**.

8.1 *Environment*

Dalam proyek profesional, *Data Warehouse* sebaiknya tidak langsung dibangun di *production*. Harus ada pemisahan lingkungan kerja agar perubahan dapat diuji tanpa mengganggu pengguna.

Lingkungan minimum:

```
Development → Testing → Staging → Production
```

Setiap lingkungan memiliki fungsi berbeda. Kesalahan fatal terjadi jika semua pekerjaan dilakukan langsung di *production*. Risiko yang muncul adalah data rusak, laporan berubah tanpa kontrol, akses bocor, dan proses ETL gagal tanpa rencana pemulihan.

A. *Development*

Development adalah lingkungan untuk membangun dan mencoba perubahan.

Digunakan untuk:

- membuat tabel baru;
- mengembangkan *ETL/ELT*;
- mencoba transformasi;
- membuat *view*;
- menguji query;
- mengembangkan *dashboard* awal;
- mengeksplorasi data contoh.

Karakteristik:

- data boleh berupa sampel;
- akses terbatas untuk tim teknis;
- error masih wajar;
- struktur sering berubah;
- tidak dipakai untuk laporan resmi.

Prinsip kritis: jangan gunakan data sensitif asli di *development* tanpa masking. Lingkungan ini sering paling longgar, sehingga risikonya tinggi.

B. *Testing*

Testing adalah lingkungan untuk menguji apakah desain, kode, dan data sudah benar.

Digunakan untuk:

- uji kualitas data;
- uji *ETL*;
- uji rekonsiliasi;
- uji performa awal;
- uji hak akses;
- uji error handling;
- uji backup dan restore terbatas.

Contoh tes:

Area	Tes
Data	null check, duplicate check, validitas kode
ETL	row count, error count, rerun test
Security	role access, masking, export restriction
Performance	query dashboard, durasi load
Recovery	restore dari backup kecil

Testing harus menghasilkan bukti, bukan hanya perasaan “sudah jalan”.

C. *Staging*

Staging adalah lingkungan yang paling mirip dengan *production*. Tujuannya adalah melakukan simulasi akhir sebelum sistem dipakai resmi.

Digunakan untuk:

- UAT;
- *dress rehearsal* deployment;
- simulasi initial load;
- simulasi historical load;
- uji performa dengan data besar;
- validasi dashboard;
- validasi akses pengguna;
- validasi dokumentasi operasi.

Karakteristik:

- skema sama dengan *production*;
- data mendekati data nyata;
- konfigurasi mendekati *production*;
- jadwal ETL diuji;
- pengguna perwakilan ikut menguji.

Prinsip kritis: *staging* adalah tempat untuk menemukan masalah sebelum pengguna menemukan masalah di *production*.

D. *Production*

Production adalah lingkungan resmi yang digunakan organisasi.

Ciri *production*:

- data menjadi rujukan resmi;
- akses dikontrol;
- backup aktif;
- monitoring aktif;
- perubahan harus melalui prosedur;
- dashboard dipakai pengguna;
- SLA berlaku;
- audit trail dijaga;
- incident response tersedia.

Di *production*, perubahan kecil pun harus hati-hati. Mengubah satu rumus KPI dapat mengubah laporan direksi. Mengubah satu *join* dapat menggandakan angka. Mengubah satu akses dapat membuka data sensitif.

Ringkasan Perbedaan *Environment*

Environment	Tujuan	Data	Pengguna	Risiko
Development	membangun dan eksperimen	sampel / masked	tim teknis	perubahan sering
Testing	pengujian teknis	data uji	QA / tim data	error terkendali
Staging	simulasi produksi	mirip nyata	tim + perwakilan user	validasi akhir
Production	operasi resmi	data resmi	pengguna organisasi	dampak bisnis tinggi

8.2 *Deployment Checklist*

Checklist deployment adalah pagar pengaman. Tujuannya memastikan sistem tidak masuk *production* dalam kondisi setengah matang.

Deployment *Data Warehouse* harus memeriksa minimal tujuh aspek:

- skema database siap;
- *ETL/ELT* sudah dites;
- hak akses benar;
- backup aktif;
- monitoring aktif;
- UAT disetujui;
- dokumentasi lengkap.

Referensi manajemen *Data Warehouse* menekankan pentingnya backup, refresh, archiving, data security, access control, data quality checks, dan performance monitoring agar *warehouse* tetap reliabel untuk pengambilan keputusan. PostgreSQL documentation juga menyediakan fasilitas monitoring database activity dan statistik aktivitas server untuk memahami performa serta perilaku database. (PostgreSQL)

A. Skema Database Siap

Yang harus dicek:

- skema `raw`, `staging`, `dw`, `mart`, `audit` sudah ada;
- tabel dimensi dibuat;
- tabel fakta dibuat;
- *primary key* dan *foreign key* sesuai;
- indeks dasar dibuat;
- *view* atau *data mart* dibuat;
- tabel log dan error tersedia;
- penamaan tabel konsisten.

Checklist:

Item	Status
skema database dibuat	
tabel staging dibuat	
tabel dimensi dibuat	
tabel fakta dibuat	
tabel audit dibuat	
indeks dibuat	
constraint diuji	
view dashboard dibuat	

Catatan kritis: jangan deploy skema yang belum memiliki tabel audit dan log. Tanpa log, masalah sulit dibuktikan.

B. *ETL/ELT* Sudah Dites

Yang harus dicek:

- proses extract berhasil;
- transformasi sesuai aturan bisnis;
- load ke dimensi berhasil;
- load ke fakta berhasil;
- error handling berjalan;
- proses bisa diulang;
- rekonsiliasi cocok;
- jadwal ETL diuji;
- performa ETL masuk SLA.

Referensi *Mastering Data Warehousing* menekankan pengujian kualitas data, validasi terhadap sistem sumber, performance tuning, UAT, deployment, dan monitoring sebagai tahap penting sebelum sistem tersedia bagi pengguna luas.

Checklist:

Item	Status
extract sukses	
transform sukses	
load dimensi sukses	
load fakta sukses	
row count dicatat	
error count dicatat	
rekonsiliasi sukses	
rerun test sukses	
jadwal ETL aktif	

C. Hak Akses Sudah Benar

Hak akses harus diuji sebelum pengguna masuk.

Yang harus dipastikan:

- role pengguna dibuat;
- akses sesuai unit dan fungsi;
- data sensitif dimasking;
- akses ekspor dibatasi;
- admin tidak memakai akun pribadi untuk proses otomatis;
- auditor memiliki akses sesuai mandat;
- pengguna tidak bisa melihat tabel mentah jika tidak perlu.

Checklist:

Role	Akses yang Diuji	Status
Direksi	dashboard ringkasan	
Manajer unit	data unit sendiri	
Analis	data mart analitik	
Auditor	data dan lineage	
Data engineer	staging dan ETL	
Admin	pengelolaan platform	

Buku referensi Snowflake menekankan pentingnya *Role-Based Access Control*, masking, audit, monitoring, dan pengujian kebijakan akses agar keamanan dan performa tetap seimbang.

D. Backup Aktif

Backup adalah syarat masuk *production*. Tidak cukup hanya membuat backup; restore juga harus diuji.

PostgreSQL documentation menjelaskan beberapa pendekatan backup seperti SQL dump, file system level backup, dan continuous archiving / point-in-time recovery. (PostgreSQL)

Checklist backup:

Item	Status
backup database aktif	
backup metadata aktif	
backup script ETL aktif	
backup konfigurasi dashboard aktif	
backup log penting aktif	
lokasi backup terpisah	
restore test dilakukan	
jadwal backup terdokumentasi	

Prinsip kritis: backup yang tidak pernah diuji restore-nya belum layak disebut backup.

E. Monitoring Aktif

Monitoring harus aktif sejak hari pertama *production*.

Yang dipantau:

- status ETL;
- durasi ETL;
- jumlah baris masuk;
- jumlah error;
- query lambat;
- storage;
- koneksi pengguna;
- akses gagal;
- dashboard lambat;
- backup gagal;
- data terlambat.

PostgreSQL documentation menjelaskan bahwa monitoring aktivitas database dapat dilakukan melalui statistik internal, alat sistem seperti `ps`, `top`, `iostat`, `vmstat`, serta analisis query lambat dengan `EXPLAIN`. (PostgreSQL)

Checklist monitoring:

Item	Status
monitoring ETL aktif	
monitoring query aktif	
monitoring storage aktif	
monitoring backup aktif	
alert error aktif	
dashboard teknis tersedia	
kontak eskalasi tersedia	

F. UAT Disetujui

UAT harus menghasilkan persetujuan bisnis.

Bukti UAT:

- skenario uji;
- hasil uji;
- daftar isu;
- perbaikan;
- sign-off pengguna;
- sign-off *data owner*;
- sign-off sponsor.

Contoh UAT:

Skenario	Ekspektasi	Status
total penjualan bulanan	sama dengan laporan resmi	
filter cabang	hanya menampilkan cabang yang dipilih	
margin produk	sesuai rumus bisnis	
akses manajer	hanya melihat unitnya	
dashboard direksi	tampil dalam SLA	

Prinsip kritis: tanpa UAT, tim teknis hanya membuktikan sistem berjalan, bukan membuktikan sistem benar menurut bisnis.

G. Dokumentasi Lengkap

Dokumentasi minimum:

- *data dictionary*;
- *ETL mapping*;
- *data lineage*;
- *access matrix*;
- *test case*;
- *deployment checklist*;
- *operational manual*;
- *rollback procedure*;
- daftar kontak darurat.

Referensi metadata menekankan bahwa metadata membantu pemahaman data, aturan transformasi, relasi, troubleshooting, dan integrasi masa depan.

8.3 Migrasi Awal

Migrasi awal adalah proses membawa data ke sistem *production* untuk pertama kali. Ini sering menjadi fase paling berisiko karena volume besar, data historis tidak bersih, dan pengguna mulai membandingkan angka lama dengan angka baru.

Migrasi awal mencakup:

```
Initial Load → Historical Load → Data Validation → Business Sign-off
```

A. *Initial Load*

Initial load adalah pemuatan data awal ke *production*. Data ini biasanya mencakup dimensi utama dan fakta awal.

Contoh:

- load seluruh pelanggan aktif;
- load seluruh produk aktif;
- load seluruh cabang;
- load transaksi tahun berjalan;
- load data proyek aktif.

Hal yang harus disiapkan:

- jendela waktu load;
- urutan load;
- kapasitas storage;
- estimasi durasi;
- backup sebelum load;
- tabel log;
- tabel error;
- validasi awal.

Urutan umum:

```
Backup awal
   ↓
Load dimensi
   ↓
Load fakta
   ↓
Load data mart
   ↓
Rekonsiliasi
   ↓
Validasi pengguna
```

B. *Historical Load*

Historical load memuat data masa lalu. Biasanya lebih sulit daripada data berjalan karena format lama tidak konsisten.

Contoh:

- transaksi 5 tahun terakhir;
- histori pelanggan;
- histori proyek;
- histori invoice;
- histori biaya;
- histori QHSE.

Risiko:

- data lama tidak lengkap;
- kode lama berbeda;
- sistem lama sudah tidak aktif;
- file historis rusak;
- definisi KPI berubah;
- data ganda;
- tidak ada pemilik data lama.

Strategi:

- load per tahun atau per bulan;
- dokumentasikan asumsi;
- gunakan tabel mapping;
- pisahkan data yang tidak valid;
- lakukan rekonsiliasi bertahap;
- minta persetujuan bisnis untuk data yang tidak sempurna.

Prinsip kritis: data historis tidak selalu bisa dibuat sempurna. Yang penting adalah **jelas batasan, asumsi, dan tingkat kepercayaannya**.

C. *Data Validation*

Validasi data memastikan hasil migrasi benar dan dapat dipercaya.

Pemeriksaan minimal:

Pemeriksaan	Tujuan
row count	jumlah baris sumber dan target sesuai
total amount	nilai transaksi cocok
duplicate check	tidak ada data ganda
null check	kolom wajib terisi
referential integrity	fakta punya dimensi valid
period coverage	periode historis lengkap
business rule	KPI sesuai definisi
sample check	pengguna memeriksa contoh data

Contoh query validasi:

```
-- Cek jumlah baris sumber dan target
     SELECT COUNT(*) AS source_rows FROM raw.sales_raw;

     SELECT COUNT(*) AS target_rows FROM dw.fact_sales;

-- Cek total nilai
     SELECT SUM(gross_amount - discount_amount) AS source_net
     FROM raw.sales_raw;

     SELECT SUM(net_amount) AS target_net
     FROM dw.fact_sales;

-- Cek fakta tanpa dimensi valid
     SELECT COUNT(*) AS missing_dimension
     FROM dw.fact_sales
     WHERE customer_key IS NULL
        OR product_key IS NULL
        OR branch_key IS NULL
        OR date_key IS NULL;
```

D. *Business Sign-off*

Business sign-off adalah persetujuan dari pemilik bisnis bahwa data hasil migrasi dapat dipakai.

Yang menyetujui:

- sponsor bisnis;
- *data owner*;
- pengguna utama;
- auditor jika relevan;
- tim data.

Isi sign-off:

Komponen	Status
KPI utama valid	
periode data sesuai	
angka cocok dengan laporan resmi	
perbedaan sudah dijelaskan	
data sensitif terlindungi	
dashboard siap dipakai	
risiko tersisa diterima	

Catatan kritis: jangan masuk *production* hanya karena tim teknis selesai. *Production* harus disetujui oleh pemilik keputusan.

8.4 Strategi *Cutover*

Cutover adalah perpindahan dari sistem lama atau proses lama ke *Data Warehouse* baru. Strategi *cutover* harus dipilih berdasarkan risiko, kompleksitas, jumlah pengguna, dan dampak bisnis.

Strategi utama:

- *big bang*;
- *parallel run*;
- *phased rollout*;
- *pilot unit*.

A. *Big Bang*

Big bang berarti seluruh pengguna berpindah ke sistem baru pada satu waktu.

Cocok untuk:

- sistem kecil;
- jumlah pengguna terbatas;
- data sudah tervalidasi kuat;
- risiko rendah;
- tidak ada sistem lama yang kompleks.

Kelebihan:

- cepat;
- tidak perlu menjalankan dua sistem lama;
- keputusan jelas.

Kekurangan:

- risiko tinggi jika ada error;
- tekanan besar pada hari peluncuran;
- rollback harus sangat siap;
- pengguna harus sudah dilatih.

Gunakan *big bang* hanya jika sistem sederhana dan risiko terkendali.

B. *Parallel Run*

Parallel run berarti sistem lama dan *Data Warehouse* baru berjalan bersamaan untuk periode tertentu.

Cocok untuk:

- laporan keuangan;
- laporan audit;
- KPI direksi;
- data yang sangat sensitif;
- organisasi yang membutuhkan kepercayaan tinggi.

Kelebihan:

- angka lama dan baru bisa dibandingkan;
- pengguna lebih percaya;
- perbedaan dapat dianalisis;
- risiko lebih rendah.

Kekurangan:

- biaya operasi lebih tinggi;
- pekerjaan ganda;
- butuh rekonsiliasi lebih banyak;
- pengguna bisa bingung jika dua sistem berbeda.

Contoh:

```
Bulan 1: laporan lama dan dashboard baru berjalan paralel
Bulan 2: selisih diselesaikan
Bulan 3: dashboard baru menjadi rujukan resmi
```

C. *Phased Rollout*

Phased rollout berarti peluncuran dilakukan bertahap.

Contoh tahapan:

```
Fase 1: Unit penjualan pusat
Fase 2: Cabang besar
Fase 3: Seluruh cabang
Fase 4: Integrasi ke domain keuangan
```

Cocok untuk:

- organisasi besar;
- banyak cabang;
- banyak unit;
- kebutuhan pelatihan bertahap;
- data kompleks.

Kelebihan:

- risiko lebih terkontrol;
- feedback dapat masuk per fase;
- tim support tidak kewalahan;
- scope lebih mudah dikelola.

Kekurangan:

- waktu lebih panjang;
- perlu koordinasi;
- versi sistem bisa berbeda antar fase.

D. *Pilot Unit*

Pilot unit berarti sistem dicoba di satu unit terlebih dahulu sebelum diperluas.

Cocok untuk:

- domain baru;
- data belum sepenuhnya matang;
- organisasi ingin validasi nyata;
- pengguna perlu adaptasi;
- desain perlu diuji di lapangan.

Contoh:

- pilot dashboard proyek di 3 proyek besar;
- pilot dashboard penjualan di 2 wilayah;
- pilot data mart keuangan di kantor pusat;
- pilot laporan SLA di satu unit layanan.

Kelebihan:

- risiko kecil;
- feedback nyata;
- cepat belajar;
- mudah diperbaiki.

Kekurangan:

- hasil pilot belum tentu mewakili seluruh organisasi;
- perlu rencana ekspansi yang jelas;
- bisa berhenti sebagai pilot jika tidak ada sponsor kuat.

Prinsip kritis: pilot bukan tempat berhenti. Pilot harus punya kriteria sukses dan rencana naik ke produksi penuh.

Ringkasan Strategi *Cutover*

Strategi	Risiko	Kecepatan	Cocok Untuk
Big bang	tinggi	cepat	sistem kecil dan matang
Parallel run	rendah-sedang	sedang	laporan kritikal
Phased rollout	sedang	bertahap	organisasi besar
Pilot unit	rendah	awal cepat	validasi desain dan adopsi

8.5 *Rollback Plan*

Rollback plan adalah rencana kembali ke kondisi aman jika deployment gagal. Tanpa rollback, deployment menjadi perjudian.

Rollback harus disiapkan sebelum *go-live*, bukan setelah insiden terjadi.

Komponen utama:

- snapshot;
- backup;
- restore procedure;
- emergency contact;
- incident response.

A. Snapshot

Snapshot adalah salinan kondisi sistem pada titik waktu tertentu.

Digunakan untuk:

- kembali ke kondisi sebelum deployment;
- membandingkan perubahan;
- memulihkan konfigurasi;
- mengurangi waktu pemulihan.

Yang perlu di-*snapshot*:

- skema database;
- data penting;
- konfigurasi ETL;
- konfigurasi dashboard;
- metadata;
- hak akses.

Catatan:

- snapshot harus diberi waktu dan versi;
- snapshot harus bisa diakses oleh tim pemulihan;
- snapshot tidak menggantikan backup jangka panjang.

B. Backup

Backup lebih formal daripada snapshot. Backup harus mengikuti kebijakan retensi dan diuji restore-nya.

Jenis backup:

- backup penuh;
- backup incremental;
- backup log;
- backup metadata;
- backup konfigurasi dashboard;
- backup script ETL.

PostgreSQL menyediakan beberapa pendekatan backup, termasuk SQL dump, backup level file system, dan continuous archiving untuk pemulihan titik waktu tertentu. (PostgreSQL)

Checklist backup sebelum deployment:

Item	Status
backup database sebelum deployment	
backup skema	
backup ETL script	
backup konfigurasi dashboard	
backup role dan permission	
backup metadata	
restore test berhasil	

C. *Restore Procedure*

Restore procedure adalah langkah tertulis untuk memulihkan sistem.

Isi minimal:

1. Hentikan akses pengguna sementara.
2. Hentikan job ETL.
3. Catat waktu dan kondisi insiden.
4. Ambil keputusan rollback.
5. Restore database dari backup/snapshot.
6. Restore konfigurasi ETL.
7. Restore dashboard.
8. Jalankan validasi data.
9. Buka akses pengguna.
10. Buat laporan insiden.

Yang harus jelas:

- siapa yang memutuskan rollback;
- siapa yang menjalankan restore;
- file backup mana yang dipakai;
- estimasi waktu pemulihan;
- cara validasi setelah restore;
- cara komunikasi ke pengguna.

D. *Emergency Contact*

Kontak darurat harus tersedia dan diuji.

Daftar minimal:

Peran	Tanggung Jawab
Incident Commander	memimpin penanganan
Data Engineer	memeriksa ETL dan data
DBA / Platform Admin	restore database
BI Admin	restore dashboard
Security Officer	cek akses dan insiden keamanan
Data Owner	validasi bisnis
Sponsor	keputusan eskalasi
Helpdesk	komunikasi pengguna

Kontak darurat harus mencakup:

- nama;
- peran;
- nomor kontak;
- kanal komunikasi;
- jam ketersediaan;
- pengganti jika tidak tersedia.

E. *Incident Response*

Incident response adalah proses menangani insiden secara terstruktur.

Jenis insiden:

- ETL gagal;
- data salah;
- dashboard tidak bisa diakses;
- query sangat lambat;
- akses data sensitif terbuka;
- data hilang;
- backup gagal;
- laporan KPI berbeda dari sumber resmi.

Langkah dasar:

```
Detect → Triage → Contain → Recover → Validate → Communicate
→ Review
```

Contoh respons:

Insiden	Respons Awal
ETL gagal	hentikan load berikutnya, cek log, jalankan ulang jika aman
data salah	karantina data, hentikan dashboard terkait
akses bocor	cabut akses, audit log, eskalasi keamanan
dashboard lambat	cek query, indeks, beban sistem
backup gagal	buat backup ulang, cek storage, eskalasi

OWASP menekankan pentingnya pencatatan keamanan dan peringatan agar organisasi dapat mendeteksi, merespons, dan memulihkan insiden secara efektif. (Apache Superset)

Prinsip kritis: insiden tidak boleh hanya “diperbaiki”. Insiden harus ditelusuri akar masalahnya, didokumentasikan, dan dicegah agar tidak berulang.

Hands-on. Membuat Checklist Deployment *Data Warehouse*

Gunakan checklist berikut sebagai dokumen kerja sebelum *go-live*. Checklist ini dapat dipakai untuk proyek sertifikasi maupun proyek organisasi.

A. Identitas Deployment

Elemen	Isi
Nama proyek	
Versi release	
Tanggal deployment	
Environment target	Production
Sponsor bisnis	
Data owner	
Penanggung jawab teknis	
Strategi cutover	Big bang / Parallel run / Phased / Pilot
Rencana rollback	Ada / Tidak

B. Checklist *Environment*

Item	Ya/Tidak	Catatan
Development selesai		
Testing selesai		
Staging sesuai production		
Production siap		
konfigurasi environment terdokumentasi		
data sensitif tidak dipakai sembarangan di non-production		

C. Checklist Skema Database

Item	Ya/Tidak	Catatan
skema `raw` tersedia		
skema `staging` tersedia		
skema `dw` tersedia		
skema `mart` tersedia		
skema `audit` tersedia		
tabel dimensi tersedia		
tabel fakta tersedia		
tabel log tersedia		
tabel error tersedia		
indeks dasar dibuat		
constraint diuji		

D. Checklist *ETL/ELT*

Item	Ya/Tidak	Catatan
extract diuji		
transform diuji		
load diuji		
incremental load diuji		
historical load diuji		
error handling diuji		
rerun aman		
jadwal ETL aktif		
row count dicatat		
error count dicatat		
rekonsiliasi dilakukan		

E. Checklist Data Migration

Item	Ya/Tidak	Catatan
initial load selesai		
historical load selesai		
data sumber dan target cocok		
data duplikat dicek		
nilai kosong dicek		
kode referensi valid		
total nilai cocok		
data mart terisi		
dashboard menampilkan data benar		

F. Checklist Security

Item	Ya/Tidak	Catatan
role pengguna dibuat		
access matrix diterapkan		
akses data sensitif dibatasi		
masking diterapkan jika perlu		
akses ekspor dibatasi		
akun admin dipisahkan		
audit log aktif		
akses diuji per role		

G. Checklist Backup dan Restore

Item	Ya/Tidak	Catatan
backup sebelum deployment dibuat		
backup database aktif		
backup metadata aktif		
backup ETL script aktif		
backup dashboard aktif		
lokasi backup aman		
restore procedure tertulis		
restore test berhasil		

H. Checklist Monitoring

Item	Ya/Tidak	Catatan
monitoring database aktif		
monitoring ETL aktif		
monitoring query lambat aktif		
monitoring storage aktif		
monitoring backup aktif		
alert error aktif		
dashboard teknis tersedia		
kontak eskalasi tersedia		

I. Checklist UAT dan *Business Sign-off*

Item	Ya/Tidak	Catatan
UAT plan tersedia		
test scenario dijalankan		
KPI utama tervalidasi		
dashboard disetujui		
akses pengguna disetujui		
selisih data dijelaskan		
data owner sign-off		
sponsor sign-off		

J. Checklist Dokumentasi

Dokumen	Ya/Tidak	Catatan
data dictionary		
ETL mapping		
data lineage		
access matrix		
test case		
deployment checklist		
operational manual		
rollback procedure		
daftar kontak darurat		

K. Checklist *Rollback Plan*

Item	Ya/Tidak	Catatan
snapshot sebelum deployment		
backup tersedia		
restore procedure tersedia		
emergency contact tersedia		
decision maker rollback jelas		
waktu maksimal rollback ditentukan		
validasi setelah rollback tersedia		
template laporan insiden tersedia		

Kesalahan Umum Deployment

Kesalahan yang sering terjadi:

- deploy langsung dari *development* ke *production*;
- tidak memakai *staging*;
- UAT hanya formalitas;
- backup dibuat tetapi restore tidak diuji;
- monitoring belum aktif saat go-live;
- hak akses tidak diuji;
- data historis dimuat tanpa validasi;
- tidak ada *business sign-off*;
- tidak ada rollback plan;
- dashboard dirilis sebelum rekonsiliasi;
- dokumentasi belum selesai;
- tidak ada kontak darurat.

Prinsip kritis: deployment gagal bukan hanya ketika sistem mati. Deployment juga gagal ketika sistem hidup, tetapi **angkanya salah, aksesnya bocor, tidak bisa diaudit, atau tidak bisa dipulihkan**.

Ringkasan Bab

Deployment *Data Warehouse* adalah proses membawa sistem dari lab ke operasi resmi. Tahap ini membutuhkan pemisahan *environment*, checklist deployment, migrasi awal, strategi *cutover*, dan *rollback plan*. Sistem hanya layak masuk *production* jika skema siap, *ETL* sudah diuji, hak akses benar, backup aktif, monitoring aktif, UAT disetujui, dan dokumentasi lengkap.

Poin penting:

- **Pisahkan *development*, *testing*, *staging*, dan *production*.**
- Jangan deploy sistem yang belum memiliki backup dan monitoring.
- Initial load dan historical load harus direkonsiliasi.
- UAT harus menghasilkan *business sign-off*.
- Pilih strategi *cutover* berdasarkan risiko.
- *Parallel run* cocok untuk laporan kritikal.
- *Phased rollout* cocok untuk organisasi besar.
- *Rollback plan* wajib disiapkan sebelum go-live.
- Insiden harus dicatat, dipulihkan, dan dievaluasi akar masalahnya.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Jelaskan perbedaan *development*, *testing*, *staging*, dan *production*.
2. Mengapa UAT harus dilakukan sebelum deployment?
3. Apa perbedaan *initial load* dan *historical load*?
4. Mengapa *business sign-off* penting sebelum go-live?
5. Jelaskan perbedaan *big bang*, *parallel run*, *phased rollout*, dan *pilot unit*.
6. Mengapa backup harus diuji restore-nya?
7. Apa saja komponen minimum dalam *rollback plan*?
8. Mengapa monitoring harus aktif sejak hari pertama *production*?

B. Studi Kasus

Sebuah perusahaan akan meluncurkan *Data Warehouse* penjualan nasional. Data historis 5 tahun sudah dimuat, dashboard direksi sudah dibuat, dan ETL harian dijadwalkan berjalan pukul 01:00. Namun, pengguna masih memakai laporan lama untuk perbandingan.

Tugas:

- pilih strategi *cutover* yang paling aman;
- buat checklist deployment minimum;
- tentukan validasi data yang wajib dilakukan;
- buat rencana backup dan restore;
- buat daftar kontak darurat;
- buat rencana rollback jika dashboard menampilkan angka salah.

C. Praktek

Buat checklist deployment untuk proyek *Data Warehouse* Anda sendiri dengan format:

Area	**Checklist Utama**	**Status**	**Catatan**
Environment			
Skema Database			
ETL/ELT			
Migrasi Data			
Security			
Backup			
Monitoring			
UAT			
Dokumentasi			
Rollback			

Bab 9. *Security*, *Privacy*, dan *Compliance Data Warehouse*

Data Warehouse menyatukan data dari banyak sistem. Karena itu, nilainya besar, tetapi resikonya juga besar. Di dalamnya bisa terdapat data pelanggan, transaksi, keuangan, proyek, vendor, pegawai, log aktivitas, hingga data personal. Jika tidak diamankan, *Data Warehouse* dapat menjadi **pusat kebocoran data**.

Keamanan *Data Warehouse* tidak cukup hanya dengan kata sandi. Sistem harus memiliki **kontrol akses, klasifikasi data, enkripsi, audit log, masking, pemisahan tugas, dan kesiapan audit**. Referensi yang digunakan menekankan bahwa keamanan dan *governance* adalah bagian penting dari pengelolaan *Data Warehouse*, termasuk akses, privasi, kepatuhan, kualitas data, dan standardisasi.

Di Indonesia, aspek privasi juga harus memperhatikan UU No. 27 Tahun 2022 tentang Pelindungan Data Pribadi. Regulasi ini mengatur jenis data pribadi, hak subjek data pribadi, pemrosesan data pribadi, kewajiban pengendali dan prosesor data pribadi, transfer data pribadi, sanksi administratif, serta ketentuan pidana terkait pelindungan data pribadi. (Database Peraturan | JDIH BPK)

Golden Rule: *Data Warehouse* yang kuat bukan hanya cepat dan lengkap, tetapi **hanya membuka data yang benar kepada orang yang benar untuk tujuan yang benar**.

9.1 Prinsip Keamanan

Keamanan *Data Warehouse* harus dirancang sejak awal. Jangan menunggu sistem selesai, lalu baru menambahkan keamanan. Jika keamanan ditempel di akhir, biasanya hasilnya tidak konsisten dan sulit diaudit.

Prinsip utama:

- *least privilege*;
- *segregation of duties*;
- *Role-Based Access Control*;
- *encryption at rest*;
- *encryption in transit*;
- *audit log*;
- *data masking*.

Referensi *Mastering Data Warehousing* menekankan bahwa *data security* mencakup langkah teknis dan organisasi untuk melindungi data dari akses tidak sah, kebocoran, dan kehilangan. Mekanisme yang disebut mencakup enkripsi, kontrol akses berbasis peran, audit rutin, masking, minimisasi data, anonimisasi, pseudonimisasi, dan kebijakan retensi data.

A. *Least Privilege*

Least privilege berarti pengguna hanya mendapat akses minimum yang diperlukan untuk menjalankan tugasnya. NIST mendefinisikan prinsip ini sebagai pembatasan hak akses pengguna atau proses hanya pada hak minimum yang diperlukan untuk menyelesaikan tugas yang ditetapkan. (NIST Computer Security Resource Center)

Contoh:

- Direksi melihat ringkasan KPI, bukan tabel mentah.
- Analis melihat *data mart*, bukan semua *raw data*.
- Tim *data engineer* dapat menulis ke tabel *staging*, tetapi tidak otomatis boleh mengekspor data pelanggan.
- Auditor mendapat akses baca dan audit trail, bukan hak mengubah data.
- Admin sistem tidak otomatis boleh membaca isi data sensitif.

Prinsip kritis: hak akses besar harus menjadi pengecualian, bukan kebiasaan.

B. *Segregation of Duties*

Segregation of duties berarti tugas kritis dipisah agar tidak satu orang menguasai seluruh proses tanpa kontrol. Tujuannya mencegah penyalahgunaan, kesalahan, dan konflik kepentingan.

Contoh pemisahan:

Fungsi	Tidak Sebaiknya Dirangkap Dengan
Pembuat ETL	Penyetuju data resmi
Admin platform	Auditor akses
Analis bisnis	Pengelola hak akses
Data owner	Operator teknis ETL
Pengguna dashboard	Pengubah rumus KPI produksi

Contoh risiko jika tidak dipisah:

- satu orang dapat mengubah rumus KPI dan menyetujui hasilnya sendiri;
- admin dapat membuka akses data sensitif tanpa persetujuan;
- analis dapat mengekspor data pelanggan tanpa audit;
- perubahan data tidak bisa ditelusuri secara independen.

Prinsip kritis: dalam sistem profesional, kepercayaan harus dibantu oleh kontrol.

C. *Role-Based Access Control*

Role-Based Access Control atau RBAC mengatur akses berdasarkan peran, bukan berdasarkan orang per orang secara acak. NIST menjelaskan RBAC sebagai pendekatan otorisasi berskala besar yang menyatukan konsep peran, hak akses, dan pengelolaan otorisasi. (NIST)

Contoh role:

- Executive
- Manager
- Analyst
- Data Engineer
- Auditor
- Admin
- Data Steward
- Security Reviewer

Contoh akses:

Role	Akses Utama
Executive	dashboard ringkasan
Manager	data unit / wilayah
Analyst	data mart analitik
Data Engineer	staging, ETL, tabel teknis
Auditor	data lineage, log, read-only
Admin	konfigurasi platform
Data Steward	metadata dan kualitas data

RBAC membuat akses lebih mudah diaudit karena pertanyaan audit menjadi jelas:

- role apa yang dimiliki pengguna?
- hak apa yang melekat pada role tersebut?
- siapa menyetujui role?
- kapan role diberikan?
- kapan role dicabut?

D. *Encryption at Rest*

Encryption at rest melindungi data saat data disimpan, misalnya di database, backup, arsip, atau object storage. NIST menjelaskan bahwa enkripsi digunakan untuk mengamankan data saat tersimpan dan saat bergerak, termasuk untuk melindungi sesi pengelolaan data tersebut. (NIST Technical Series)

Data yang perlu diperhatikan:

- tabel pelanggan;
- data transaksi;
- data keuangan;
- data personal;
- backup database;
- file ekspor;
- arsip historis;
- log yang berisi data sensitif.

Kontrol minimal:

- enkripsi storage;
- enkripsi backup;
- pengelolaan kunci;
- pembatasan akses kunci;
- rotasi kunci sesuai kebijakan;
- dokumentasi siapa yang dapat membuka data.

Catatan kritis: backup juga harus dienkripsi. Banyak kebocoran terjadi bukan dari database utama, tetapi dari file backup yang tidak terlindungi.

E. *Encryption in Transit*

Encryption in transit melindungi data saat bergerak melalui jaringan, misalnya dari sumber data ke *staging*, dari *warehouse* ke dashboard, atau dari API ke pipeline.

Risiko jika tidak dienkripsi:

- data dapat disadap;
- kredensial dapat bocor;
- data sensitif dapat terlihat di jaringan;
- integritas data sulit dijamin.

Kontrol minimal:

- koneksi database terenkripsi;
- API melalui kanal aman;
- sertifikat valid;
- tidak mengirim file sensitif melalui kanal tidak aman;
- pembatasan akses jaringan;
- monitoring koneksi gagal atau mencurigakan.

F. *Audit Log*

Audit log mencatat aktivitas penting di *Data Warehouse*. Tanpa log, organisasi sulit menjawab siapa mengakses apa, kapan, dari mana, dan apa yang berubah.

OWASP Logging Cheat Sheet menekankan pentingnya mekanisme logging aplikasi, terutama untuk *security logging*. OWASP juga menekankan bahwa transaksi harus memiliki audit trail dengan kontrol integritas agar log tidak mudah diubah atau dihapus. (OWASP Cheat Sheet Series)

Log penting:

- login berhasil dan gagal;
- akses data sensitif;
- perubahan role;
- perubahan hak akses;
- query ekspor besar;
- perubahan tabel produksi;
- perubahan ETL;
- perubahan rumus KPI;
- kegagalan backup;
- kegagalan ETL;
- anomali akses.

Contoh struktur log:

Waktu	User	Role	Aksi	Objek	Status	IP / Lokasi
2026-01-01 08:00	analyst01	Analyst	SELECT	mart_sales	success	internal
2026-01-01 08:05	user02	Manager	EXPORT	customer_data	denied	internal
2026-01-01 08:10	admin01	Admin	GRANT ROLE	Analyst	success	internal

Prinsip kritis: log harus dilindungi. Log yang bisa diubah bebas oleh admin yang sama akan lemah sebagai bukti audit.

G. *Data Masking*

Data masking menyamarkan data sensitif agar pengguna tetap bisa bekerja tanpa melihat nilai asli.

Contoh:

Data Asli	**Data Dimasking**
081234567890	0812******90
nama@email.com	n***@email.com
1234567890123456	**********3456
Jl. Merdeka No. 10	Jl. Merdeka ****

Gunakan masking untuk:

- data pelanggan;
- data pegawai;
- nomor identitas;
- nomor rekening;
- kontak pribadi;
- alamat lengkap;
- data keuangan pribadi;
- data kesehatan jika ada;
- data anak jika ada.

Referensi yang digunakan menjelaskan bahwa *data masking* dapat digunakan pada dataset untuk *development*, pengujian, atau analisis agar data tertentu disamarkan, tetapi data tetap dapat digunakan secara fungsional.

9.2 Klasifikasi Data

Klasifikasi data membantu organisasi menentukan tingkat perlindungan. Tidak semua data memiliki risiko yang sama. Data publik berbeda dari data personal. Data ringkasan berbeda dari data transaksi detail. Data produksi berbeda dari data uji.

Klasifikasi minimal:

- Public
- Internal
- Confidential
- Restricted
- Personal data

Referensi desain *Data Warehouse* menekankan bahwa akses kontrol dan enkripsi perlu diterapkan untuk melindungi data sensitif seperti informasi pelanggan dan data keuangan, terutama agar sesuai dengan regulasi privasi.

A. Public

Public adalah data yang boleh dibuka kepada publik.

Contoh:

- laporan tahunan yang sudah dipublikasikan;
- informasi produk umum;
- daftar layanan publik;
- statistik yang sudah dianonimkan;
- metadata non-sensitif yang disetujui.

Kontrol:

- tetap perlu validasi;
- tidak boleh mengandung data personal;
- harus disetujui sebelum dipublikasikan;
- versi publik harus jelas.

B. Internal

Internal adalah data untuk penggunaan internal organisasi.

Contoh:

- laporan operasional internal;
- struktur organisasi;
- daftar cabang;
- ringkasan penjualan internal;
- status proyek umum;
- data inventaris non-sensitif.

Kontrol:

- hanya untuk pegawai / pihak berwenang;
- tidak boleh dibagikan keluar tanpa persetujuan;
- akses berdasarkan role;
- ekspor dibatasi jika perlu.

C. Confidential

Confidential adalah data rahasia yang jika bocor dapat merugikan organisasi.

Contoh:

- data transaksi detail;
- data margin;
- data vendor;
- kontrak;
- strategi penjualan;
- data biaya;
- data proyek sensitif;
- data audit internal.

Kontrol:

- akses terbatas;
- enkripsi;
- audit log;
- persetujuan akses;
- pembatasan ekspor;
- masking jika memungkinkan.

D. Restricted

Restricted adalah data sangat sensitif yang aksesnya harus sangat dibatasi.

Contoh:

- kredensial sistem;
- kunci enkripsi;
- data personal sensitif;
- data keuangan pribadi;
- data investigasi;
- data keamanan;
- data audit kritikal;
- data yang terkena batasan regulasi.

Kontrol:

- akses sangat terbatas;
- persetujuan berlapis;
- log wajib;
- enkripsi wajib;
- masking / tokenisasi;
- review akses berkala;
- larangan ekspor kecuali disetujui.

E. *Personal Data*

Personal data adalah data yang terkait dengan individu. UU PDP Indonesia menyatakan bahwa data pribadi terdiri atas data pribadi yang bersifat spesifik dan umum; data spesifik mencakup antara lain data kesehatan, biometrik, genetika, catatan kejahatan, data anak, dan data keuangan pribadi. (Database Peraturan | JDIH BPK)

Contoh data personal:

- nama;
- nomor identitas;
- alamat;
- nomor telepon;
- email pribadi;
- data keuangan pribadi;
- data pegawai;
- data pelanggan individu;
- data biometrik jika ada;
- data kesehatan jika ada.

Kontrol:

- dasar pemrosesan jelas;
- akses terbatas;
- masking;
- minimisasi data;
- retensi jelas;
- audit akses;
- penghapusan sesuai kebijakan;
- perlindungan saat ekspor;
- perlindungan saat backup.

Prinsip kritis: data personal tidak boleh masuk ke *Data Warehouse* hanya karena “mungkin berguna”. Harus jelas tujuan, dasar, akses, retensi, dan kontrolnya.

9.3 *Access Matrix*

Access matrix adalah tabel yang menjelaskan hak akses tiap role terhadap data dan fungsi sistem. Ini adalah dokumen wajib untuk *Data Warehouse* profesional.

Tujuannya:

- mencegah akses berlebihan;
- memudahkan audit;
- memperjelas tanggung jawab;
- mempercepat onboarding pengguna;
- memudahkan pencabutan akses;
- mendukung prinsip *least privilege*.

Contoh *Access Matrix* Dasar

Role	Read	Write	Admin	Export	Sensitive Data
Executive	Ya	Tidak	Tidak	Terbatas	Masked
Analyst	Ya	Tidak	Tidak	Ya	Masked
Engineer	Ya	Ya	Tidak	Tidak	Terbatas
Admin	Ya	Ya	Ya	Tidak	Terbatas

Tabel ini masih perlu diperjelas berdasarkan jenis data. Misalnya, “read” untuk Executive biasanya bukan berarti membaca semua tabel mentah. Executive seharusnya membaca dashboard ringkasan dan *data mart* resmi.

Access Matrix Lebih Detail

Role	Raw Data	Staging	DW Core	Data Mart	Dashboard	Export	Data Personal
Executive	Tidak	Tidak	Tidak	Ringkasan	Ya	Terbatas	Masked
Manager	Tidak	Tidak	Terbatas	Unit sendiri	Ya	Terbatas	Masked
Analyst	Tidak	Tidak	Terbatas	Ya	Ya	Ya, terbatas	Masked
Data Engineer	Ya	Ya	Ya	Ya	Terbatas	Tidak	Terbatas
Data Steward	Terbatas	Terbatas	Terbatas	Ya	Ya	Terbatas	Masked
Auditor	Terbatas	Terbatas	Ya, read-only	Ya	Ya	Terbatas	Sesuai mandat
Admin	Terbatas	Terbatas	Terbatas	Terbatas	Terbatas	Tidak	Terbatas

Aturan Tambahan untuk *Access Matrix*

Agar matriks akses tidak hanya menjadi tabel formalitas, tambahkan aturan berikut:

- setiap role harus punya pemilik;
- setiap akses sensitif harus punya pemberi persetujuan;
- akses sementara harus punya tanggal kedaluwarsa;
- ekspor data harus dicatat;
- akses admin harus direview berkala;
- akses pegawai keluar harus dicabut segera;
- akses data personal harus dibatasi berdasarkan kebutuhan kerja;
- akses *raw data* hanya untuk tim yang benar-benar membutuhkan.

Prinsip kritis: akses yang tidak direview akan selalu membesar dari waktu ke waktu. Ini disebut *access creep* dan berbahaya untuk audit.

Contoh Kontrol SQL Sederhana

Contoh konsep role:

```
CREATE ROLE dw_executive;
CREATE ROLE dw_analyst;
CREATE ROLE dw_engineer;
CREATE ROLE dw_auditor;
```

Contoh akses ringkasan untuk Executive:

```
GRANT SELECT ON mart.sales_summary_monthly TO dw_executive;
```

Contoh akses analitik untuk Analyst:

```
GRANT SELECT ON mart.sales_summary_monthly TO dw_analyst;
GRANT SELECT ON mart.customer_segment_summary TO dw_analyst;
```

Contoh akses teknis untuk Engineer:

```
GRANT SELECT, INSERT, UPDATE ON staging.sales_clean TO
dw_engineer;
GRANT SELECT, INSERT ON dw.fact_sales TO dw_engineer;
```

Contoh pembatasan:

```
REVOKE ALL ON raw.customer_raw FROM dw_analyst;
REVOKE ALL ON raw.customer_raw FROM dw_executive;
```

Catatan: contoh di atas adalah pola dasar. Implementasi produksi harus menyesuaikan platform, struktur role, kebijakan keamanan, dan regulasi organisasi.

9.4 *Audit Readiness*

Audit readiness berarti *Data Warehouse* siap diperiksa kapan pun. Audit bukan hanya mencari kesalahan. Audit memastikan data dapat dipercaya, aman, dan dikelola sesuai aturan.

Pertanyaan audit utama:

- siapa akses data?
- kapan data berubah?
- dari mana data berasal?
- siapa menyetujui perubahan?
- apakah data sensitif terlindungi?

Metadata sangat membantu audit karena mencatat asal data, transformasi, aturan bisnis, *data lineage*, kepemilikan, akses, dan pola penggunaan. Referensi yang digunakan menjelaskan bahwa metadata mendukung *governance*, compliance, data quality, dokumentasi, system integration, dan visibilitas terhadap ownership, access restrictions, serta usage patterns.

A. Siapa Akses Data?

Audit harus bisa menjawab:

- user siapa?
- role apa?
- data apa yang diakses?
- kapan?
- dari mana?
- apakah aksesnya sah?
- apakah data diekspor?
- apakah akses tersebut masih diperlukan?

Log minimal:

Kolom Log	Fungsi
event_time	waktu kejadian
user_id	identitas pengguna
role_name	role saat akses
action	SELECT, INSERT, UPDATE, EXPORT
object_name	tabel, view, dashboard
status	success / failed / denied
source_ip	asal akses
query_id	ID query jika tersedia

B. Kapan Data Berubah?

Audit harus bisa melihat perubahan data dan perubahan struktur.

Yang perlu dicatat:

- perubahan tabel;
- perubahan *view*;
- perubahan rumus KPI;
- perubahan ETL;
- perubahan data master;
- perubahan role;
- perubahan akses;
- perubahan jadwal refresh;
- perubahan dashboard resmi.

Contoh log perubahan:

Waktu	Objek	Perubahan	Oleh	Disetujui Oleh
2026-02-01	KPI Margin	rumus margin diperbarui	data_engineer01	Data Owner
2026-02-03	DimBranch	tambah cabang baru	data_steward01	Operasi
2026-02-05	Access Role	tambah analyst baru	admin01	Manager Data

C. Dari Mana Data Berasal?

Pertanyaan ini dijawab melalui *data lineage*.

Contoh:

```
sales.csv
   ↓
raw.sales_raw
   ↓
staging.sales_clean
   ↓
dw.fact_sales
   ↓
mart.sales_summary_monthly
   ↓
dashboard_penjualan
```

Audit harus bisa mengetahui:

- sumber data awal;
- waktu ekstraksi;
- aturan transformasi;
- tabel target;
- siapa pemilik data;
- apakah ada data yang ditolak;
- apakah rekonsiliasi berhasil.

Referensi metadata repository menekankan bahwa metadata mendokumentasikan sumber data, aturan translasi, area pengguna, serta peta dari data sumber ke target.

D. Siapa Menyetujui Perubahan?

Perubahan di *Data Warehouse* tidak boleh hanya keputusan teknis. Perubahan yang mempengaruhi KPI, akses, data sensitif, dan dashboard resmi harus disetujui.

Contoh perubahan yang perlu persetujuan:

- rumus KPI;
- definisi pelanggan aktif;
- perubahan akses data sensitif;
- penambahan sumber data baru;
- perubahan frekuensi refresh;
- penghapusan data historis;
- perubahan retensi;
- perubahan dashboard direksi.

Dokumen persetujuan minimal:

Perubahan	Dampak	Pemohon	Penyetuju	Tanggal	Status

E. Apakah Data Sensitif Terlindungi?

Audit harus bisa membuktikan:

- data sensitif sudah diklasifikasi;
- akses terbatas;
- masking diterapkan;
- enkripsi aktif;
- ekspor dikontrol;
- log akses tersedia;
- backup terlindungi;
- retensi sesuai kebijakan;
- data tidak disimpan lebih lama dari kebutuhan.

Prinsip kritis: audit bukan hanya soal “ada dokumen”. Audit harus bisa membuktikan kontrol benar-benar berjalan.

Hands-on. Membuat Matriks Akses dan Klasifikasi Data

Latihan ini membantu peserta membuat kontrol keamanan awal untuk *Data Warehouse*.

Langkah 1 — Identifikasi Domain Data

Contoh domain: **Data Warehouse Penjualan**.

Sumber data:

- transaksi penjualan;
- pelanggan;
- produk;
- cabang;
- invoice;
- pembayaran;
- target penjualan.

Langkah 2 — Klasifikasikan Data

Dataset	Contoh Isi	Klasifikasi	Alasan
DimProduct	kode produk, nama produk, kategori	Internal	data bisnis internal
DimBranch	kode cabang, lokasi cabang	Internal	data operasional
FactSales	transaksi, nilai penjualan, margin	Confidential	berdampak bisnis jika bocor
DimCustomer	nama pelanggan, kontak, segmentasi	Personal / Confidential	mengandung data pelanggan
FactInvoice	invoice, nilai, status pembayaran	Confidential	data keuangan
AuditLog	user, waktu akses, objek	Restricted	bukti audit dan keamanan
RawCustomer	data pelanggan mentah	Restricted	data mentah berisiko tinggi

Langkah 3 — Tentukan Role

Role	Fungsi
Executive	melihat KPI ringkasan
Manager	melihat data unit / wilayah
Analyst	analisis data mart
Data Engineer	membangun dan menjalankan ETL
Data Steward	menjaga definisi dan kualitas data
Auditor	memeriksa data, log, dan lineage
Admin	mengelola platform dan akses teknis

Langkah 4 — Buat *Access Matrix*

Role	Public	Internal	Confidential	Restricted	Personal Data	Export
Executive	Read	Read	Ringkasan	Tidak	Masked	Terbatas
Manager	Read	Read	Unit sendiri	Tidak	Masked	Terbatas
Analyst	Read	Read	Read terbatas	Tidak	Masked	Ya, terbatas
Data Engineer	Read	Read / Write	Read/Write terbatas	Terbatas	Terbatas	Tidak
Data Steward	Read	Read	Read terbatas	Tidak	Masked	Terbatas
Auditor	Read	Read	Read-only	Read-only sesuai mandat	Sesuai mandat	Terbatas
Admin	Terbatas	Terbatas	Terbatas	Terbatas	Terbatas	Tidak

Langkah 5 — Tentukan Kontrol Tiap Klasifikasi

Klasifikasi	Kontrol Minimum
Public	persetujuan publikasi, validasi isi
Internal	role-based access, tidak dibagikan keluar
Confidential	enkripsi, audit log, pembatasan ekspor
Restricted	persetujuan berlapis, log wajib, akses sangat terbatas
Personal Data	masking, minimisasi, retensi, audit akses

Langkah 6 — Buat Checklist Audit

Pertanyaan Audit	Bukti
Siapa yang bisa mengakses data pelanggan?	access matrix
Kapan user terakhir mengakses data sensitif?	audit log
Apakah data pelanggan dimasking untuk analis?	contoh view / query
Siapa menyetujui akses analyst?	approval record
Dari mana data penjualan berasal?	data lineage
Apakah backup terenkripsi?	konfigurasi backup
Apakah ekspor data dicatat?	export log
Apakah akses pegawai keluar sudah dicabut?	access review

Langkah 7 — Contoh *Policy* Singkat

Contoh kebijakan ringkas untuk *Data Warehouse*:

1. Semua akses Data Warehouse harus berbasis role.
2. Akses ke data personal harus menggunakan prinsip least privilege.
3. Data personal untuk analyst wajib dimasking.
4. Ekspor data confidential harus dicatat dalam audit log.
5. Perubahan rumus KPI harus disetujui data owner.
6. Backup database dan file ekspor sensitif harus dienkripsi.
7. Akses admin harus direview minimal setiap kuartal.
8. Data raw hanya boleh diakses oleh tim yang diberi mandat teknis.
9. Data lineage wajib tersedia untuk laporan resmi.
10. Semua insiden akses tidak sah harus dicatat dan eskalasi.

Checklist Bab 9 untuk Sertifikasi

Kompetensi	Harus Bisa
Least privilege	menjelaskan akses minimum sesuai tugas
Segregation of duties	memisahkan tugas teknis, bisnis, audit
RBAC	membuat role dan hak akses
Enkripsi	membedakan at rest dan in transit
Audit log	menentukan event yang wajib dicatat
Data masking	menentukan data yang harus disamarkan
Klasifikasi data	membedakan Public, Internal, Confidential, Restricted, Personal
Access matrix	membuat tabel akses berdasarkan role
Audit readiness	menjawab siapa akses data, kapan berubah, dari mana asalnya
Compliance	menjelaskan kaitan DW dengan regulasi privasi

Kesalahan Umum

Kesalahan yang sering terjadi:

- semua pengguna diberi akses terlalu luas;
- analis dapat membaca data mentah berisi data personal;
- tidak ada klasifikasi data;
- tidak ada masking;
- backup tidak dienkripsi;
- log tidak lengkap;
- log dapat dihapus oleh admin biasa;
- akses mantan pegawai tidak dicabut;
- perubahan KPI tidak disetujui *data owner*;
- data personal disimpan tanpa tujuan jelas;
- ekspor data tidak dicatat;
- UAT tidak menguji akses;
- audit baru dipikirkan setelah insiden.

Prinsip kritis: keamanan yang baik tidak menghambat analitik. Keamanan yang baik membuat analitik **aman, legal, dan dipercaya**.

Ringkasan Bab

Security, *privacy*, dan *compliance* adalah fondasi *Data Warehouse* profesional. Karena *Data Warehouse* menggabungkan banyak sumber data, ia harus memiliki kontrol yang lebih kuat daripada sistem biasa. Keamanan harus mencakup *least privilege*, *segregation of duties*, RBAC, enkripsi, audit log, masking, klasifikasi data, dan kesiapan audit.

Poin penting:

- **Jangan beri akses berdasarkan jabatan semata; beri akses berdasarkan kebutuhan kerja.**
- Data harus diklasifikasikan sebelum diberi akses.
- Data personal harus diproses dengan kontrol khusus.
- RBAC memudahkan kontrol dan audit.
- Enkripsi wajib dipikirkan untuk data tersimpan dan data bergerak.
- Audit log harus menjawab siapa, kapan, apa, dari mana, dan hasilnya.
- *Data masking* mengurangi risiko paparan data sensitif.
- *Access matrix* adalah dokumen kerja penting untuk sertifikasi dan audit.
- *Audit readiness* harus dibangun sejak desain, bukan setelah sistem berjalan.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Jelaskan prinsip *least privilege* dalam konteks *Data Warehouse*.
2. Mengapa *segregation of duties* penting untuk mencegah konflik kepentingan?
3. Apa kelebihan RBAC dibanding pemberian akses per orang tanpa standar?
4. Jelaskan perbedaan *encryption at rest* dan *encryption in transit*.
5. Sebutkan minimal 7 aktivitas yang harus masuk *audit log*.
6. Mengapa *data masking* penting untuk data personal?
7. Apa perbedaan *Confidential* dan *Restricted*?
8. Bagaimana *data lineage* membantu audit?

B. Studi Kasus

Sebuah organisasi memiliki *Data Warehouse* penjualan yang berisi transaksi, pelanggan, cabang, produk, invoice, dan log akses. Direksi membutuhkan dashboard ringkasan. Analis membutuhkan data untuk eksplorasi. Auditor membutuhkan bukti asal data dan log. Tim *data engineer* membutuhkan akses teknis untuk ETL.

Tugas:

- klasifikasikan setiap dataset;
- buat role minimal 5 jenis;
- buat *access matrix*;
- tentukan data yang harus di masking;
- tentukan event yang harus dicatat dalam audit log;
- tentukan siapa yang menyetujui akses ke data sensitif.

C. Praktik

Buat matriks berikut untuk organisasi Anda:

Dataset	Klasifikasi	Data Owner	Role yang Boleh Akses	Masking	Export	Audit Log

BAB 10. Performance Tuning dan Capacity Planning

Data Warehouse yang benar secara desain belum tentu cepat. Semakin besar volume data, semakin banyak pengguna, semakin kompleks *query*, dan semakin sering proses *refresh*, maka risiko performa akan meningkat. Masalah ini tidak boleh ditangani setelah sistem lambat. Ia harus dipikirkan sejak desain.

Performance tuning adalah proses mengoptimalkan *query*, struktur tabel, indeks, partisi, agregasi, *materialized view*, *ETL*, dan sumber daya komputasi. *Capacity planning* adalah proses menghitung kebutuhan storage, compute, backup, retensi, pertumbuhan data, dan beban pengguna.

Referensi yang digunakan menekankan bahwa strategi peningkatan performa *Data Warehouse* mencakup *data partitioning*, *indexing*, *materialized views*, denormalisasi terkendali, optimasi sumber daya, kompresi, arsip data, optimasi *query*, caching, dan *parallel processing*. Referensi lain juga menyebut bahwa tuning mencakup *query optimization*, *indexing*, *partitioning*, optimasi infrastruktur, kompresi, agregasi, optimasi *ETL*, caching, dan *workload management*.

Golden Rule: performa *Data Warehouse* tidak diukur dari seberapa besar servernya, tetapi dari **seberapa cepat pengguna mendapat jawaban yang benar dengan biaya yang terkendali**.

10.1 Faktor Performa

Performa *Data Warehouse* dipengaruhi banyak faktor. Tidak ada satu teknik yang menyelesaikan semua masalah. Indeks dapat mempercepat *query*, tetapi memperlambat *load*. Partisi dapat mempercepat pencarian data historis, tetapi menambah kompleksitas pengelolaan. *Materialized view* dapat mempercepat laporan, tetapi membutuhkan proses *refresh*.

Faktor utama:

- volume data;
- jumlah user;
- kompleksitas *query*;
- *refresh frequency*;
- *index*;
- *partitioning*;
- agregasi;
- *materialized view*;
- *concurrency*.

A. Volume Data

Volume data adalah faktor paling jelas. Semakin besar tabel fakta, semakin besar biaya baca, penyimpanan, backup, dan pemrosesan.

Contoh pertumbuhan:

Domain	Potensi Volume
Transaksi penjualan	jutaan baris per bulan
Log aplikasi	jutaan baris per hari
Sensor energi	ribuan baris per jam
Progres proyek	ribuan baris per minggu
Invoice	ratusan ribu baris per bulan

Namun volume besar tidak selalu lambat jika desainnya baik. Masalah biasanya muncul ketika:

- semua data dibaca tanpa filter;
- tabel fakta tidak dipartisi;
- *query* tidak memakai indeks;
- agregasi selalu dihitung ulang;
- data historis lama tetap aktif;
- dashboard membaca data detail langsung.

Prinsip kritis: data besar harus diatur, bukan hanya disimpan.

B. Jumlah User

Jumlah user mempengaruhi beban sistem. Satu analis yang menjalankan *query* berat bisa lebih membebani sistem daripada 100 pengguna dashboard ringan.

Jenis pengguna:

Pengguna	Karakter Beban
Executive	dashboard ringkas, sering membaca agregat
Manager	filter unit / wilayah, laporan periodik
Analyst	ad hoc query, eksplorasi, join kompleks
Data Engineer	ETL, load, transformasi
Auditor	query histories, lineage, sampling
Dashboard publik internal	banyak akses bersamaan

Yang harus dihitung bukan hanya jumlah user total, tetapi **user aktif bersamaan**.

Contoh:

```
Total user terdaftar = 500
User aktif harian = 120
User aktif bersamaan jam sibuk = 30
User query berat = 5
```

C. Kompleksitas *Query*

Query sederhana bisa cepat walaupun data besar. *Query* kompleks bisa lambat walaupun data tidak terlalu besar.

Faktor kompleksitas:

- banyak *join*;
- agregasi besar;
- *subquery* bertingkat;
- filter tidak selektif;
- fungsi pada kolom filter;
- sort besar;
- *distinct* pada data besar;
- *window function* berat;
- membaca terlalu banyak kolom;
- tidak memanfaatkan partisi.

Contoh buruk:

```
SELECT *
FROM fact_sales
WHERE EXTRACT(YEAR FROM sales_date) = 2026;
```

Masalah: fungsi pada kolom tanggal dapat membuat optimasi lebih sulit.

Lebih baik:

```
SELECT
    date_key,
    product_key,
    SUM(net_amount) AS total_sales
FROM fact_sales
WHERE sales_date >= DATE '2026-01-01'
  AND sales_date < DATE '2027-01-01'
GROUP BY date_key, product_key;
```

PostgreSQL menyediakan `EXPLAIN` untuk menampilkan rencana eksekusi dan `EXPLAIN ANALYZE` untuk menjalankan perintah serta mengukur biaya waktu nyata, meskipun `EXPLAIN ANALYZE` dapat menambah overhead saat pengukuran. (PostgreSQL)

D. *Refresh Frequency*

Semakin sering data diperbarui, semakin tinggi beban sistem.

Pilihan *refresh*:

Frekuensi	Cocok Untuk	Risiko
Bulanan	laporan strategis	data tidak aktual
Mingguan	proyek, manajemen	cukup ringan
Harian	dashboard eksekutif	butuh jadwal stabil
Per jam	operasional	beban lebih tinggi
Near-real-time	alert kritikal	kompleks dan mahal

Pertanyaan kritis:

- apakah keputusan benar-benar membutuhkan data real-time?
- apakah sistem sumber siap?
- apakah ETL mampu?
- apakah dashboard perlu membaca data detail?
- apakah biaya tambahan sebanding dengan nilai bisnis?

Prinsip kritis: jangan memilih *near-real-time* untuk keputusan yang hanya dibuat bulanan.

E. *Index*

Index mempercepat pencarian data pada kolom yang sering dipakai untuk filter, join, atau sort. Referensi yang digunakan menjelaskan bahwa indeks berperan penting untuk optimasi *query* karena membuat sistem dapat menemukan baris tertentu lebih efisien dan mengurangi jumlah data yang harus dipindai.

Contoh kolom yang sering diindeks:

- `date_key`;
- `customer_key`;
- `product_key`;
- `branch_key`;
- `invoice_number`;
- `project_key`;
- kombinasi `date_key, product_key`;
- kombinasi `date_key, branch_key`.

Namun indeks tidak gratis. Terlalu banyak indeks dapat:

- memperlambat proses *load*;
- memperbesar storage;
- menambah biaya maintenance;
- memperlambat update;
- membuat desain lebih kompleks.

Prinsip kritis: indeks harus dibuat berdasarkan pola *query*, bukan berdasarkan semua kolom yang terlihat penting.

F. *Partitioning*

Partitioning membagi tabel besar menjadi bagian lebih kecil berdasarkan kriteria tertentu, misalnya tanggal, range, hash, atau wilayah. PostgreSQL documentation menjelaskan bahwa tabel dapat dideklarasikan sebagai tabel terpartisi, dengan kunci partisi berupa kolom atau ekspresi tertentu. (PostgreSQL)

Contoh partisi berdasarkan bulan:

```
fact_sales_2026_01
fact_sales_2026_02
fact_sales_2026_03
```

Manfaat:

- *query* historis lebih cepat;
- data lama mudah diarsipkan;
- maintenance lebih mudah;
- load per periode lebih terkontrol;
- backup dan retensi lebih fleksibel.

Referensi yang digunakan menjelaskan bahwa *partitioning* membagi tabel besar menjadi bagian yang lebih kecil dan mudah dikelola, memungkinkan akses dan pemrosesan paralel, mengurangi data yang dipindai, dan mempercepat eksekusi *query*.

G. Agregasi

Agregasi menyimpan ringkasan data agar dashboard tidak selalu menghitung dari data detail.

Contoh:

Detail	Agregasi
transaksi per item	penjualan per bulan
pemakaian energi per jam	energi per hari
progres proyek mingguan	progres bulanan
invoice detail	total invoice per pelanggan
log akses	jumlah akses per jam

Contoh tabel agregat:

```
mart_sales_monthly
mart_sales_by_branch
mart_sales_by_product
mart_project_progress_monthly
```

Agregasi cocok untuk:

- dashboard eksekutif;
- laporan bulanan;
- KPI yang sering dipakai;
- *query* berat yang berulang.

Risiko:

- data agregat harus di-*refresh*;
- definisi agregat harus jelas;
- angka bisa berbeda jika detail berubah;
- perlu rekonsiliasi.

H. *Materialized View*

Materialized view menyimpan hasil *query* sebagai objek fisik seperti tabel. Ini berbeda dari *view* biasa yang dihitung ulang saat dipanggil. PostgreSQL documentation menjelaskan bahwa *materialized views* menyimpan hasil *query* dalam bentuk seperti tabel, sehingga hasilnya dapat dipakai kembali tanpa menghitung ulang setiap saat. (PostgreSQL)

Contoh:

```
CREATE MATERIALIZED VIEW mart.mv_sales_monthly AS
SELECT
    d.year_number,
    d.month_number,
    b.region_name,
    SUM(f.net_amount) AS total_sales,
    SUM(f.margin_amount) AS total_margin
FROM dw.fact_sales f
JOIN dw.dim_date d ON f.date_key = d.date_key
JOIN dw.dim_branch b ON f.branch_key = b.branch_key
GROUP BY d.year_number, d.month_number, b.region_name;
```

Manfaat:

- mempercepat laporan yang sering dipakai;
- mengurangi beban *query* kompleks;
- cocok untuk agregasi dan join berat.

Risiko:

- butuh storage tambahan;
- perlu jadwal *refresh*;
- data bisa tidak terbaru jika belum di-*refresh*;
- perlu kontrol rekonsiliasi.

Referensi yang digunakan juga menjelaskan bahwa *materialized views* menyimpan hasil *query* kompleks yang sudah dihitung sebelumnya sehingga mempercepat laporan, terutama untuk agregasi dan join kompleks.

I. *Concurrency*

Concurrency adalah kemampuan sistem menangani banyak *query* atau proses secara bersamaan.

Masalah *concurrency* muncul ketika:

- banyak dashboard dibuka bersamaan;
- ETL berjalan saat jam kerja;
- analis menjalankan query berat;
- laporan bulanan dibuka banyak pengguna;
- proses backup bersamaan dengan *load*;
- sistem tidak memisahkan workload.

Dokumentasi Snowflake menjelaskan bahwa jumlah *query* yang dapat diproses secara bersamaan bergantung pada ukuran dan kompleksitas *query*; jika sumber daya tidak cukup, *query* akan masuk antrean. (Dokumentasi Snowflake) Dokumentasi Snowflake juga menyebut bahwa *multi-cluster warehouses* dapat digunakan untuk menskalakan compute sesuai kebutuhan user dan *query concurrency* pada jam sibuk maupun sepi. (Dokumentasi Snowflake)

Prinsip kritis: jangan biarkan ETL, dashboard eksekutif, dan query eksploratif berat berebut sumber daya yang sama tanpa prioritas.

10.2 *Indexing* dan *Partitioning*

Indexing dan *partitioning* adalah dua teknik paling penting dalam optimasi fisik *Data Warehouse*. Keduanya harus digunakan dengan disiplin.

Referensi yang digunakan menjelaskan bahwa *indexing* mempercepat pengambilan data pada kolom yang sering dipakai untuk filter atau join, sedangkan *partitioning* membagi tabel besar menjadi bagian lebih kecil berdasarkan kriteria seperti waktu, range, atau hash untuk meningkatkan performa dan pengelolaan.

A. Strategi *Indexing*

Gunakan indeks pada kolom yang sering dipakai untuk:

- filter;
- join;
- sort;
- grouping;
- pencarian data spesifik;
- relasi fakta dan dimensi.

Contoh:

```
CREATE INDEX idx_fact_sales_date
ON dw.fact_sales(date_key);

CREATE INDEX idx_fact_sales_product
ON dw.fact_sales(product_key);

CREATE INDEX idx_fact_sales_branch_date
ON dw.fact_sales(branch_key, date_key);
```

Pola pemilihan indeks:

Pola Query	Indeks yang Disarankan
filter tanggal	date_key
filter produk	product_key
laporan cabang per bulan	(branch_key, date_key)
join fakta ke pelanggan	customer_key
invoice lookup	invoice_number

Namun, hindari membuat indeks untuk semua kolom. Referensi *Mastering Data Warehousing* mengingatkan bahwa indeks mempercepat retrieval tetapi membutuhkan storage tambahan dan dapat memperlambat insert karena ada overhead maintenance.

B. Kapan Indeks Tidak Membantu?

Indeks tidak selalu mempercepat.

Indeks kurang efektif jika:

- tabel sangat kecil;
- *query* membaca sebagian besar tabel;
- kolom punya nilai sangat sedikit;
- filter tidak selektif;
- fungsi diterapkan pada kolom filter;
- statistik database tidak diperbarui;
- indeks tidak sesuai pola query.

Contoh tidak ideal:

```
WHERE UPPER(branch_code) = 'JKT01'
```

Lebih baik simpan kode cabang dalam format standar sejak awal:

```
WHERE branch_code = 'JKT01'
```

PostgreSQL memiliki perintah `ANALYZE` untuk mengumpulkan statistik isi tabel agar perencana *query* dapat membuat keputusan eksekusi yang lebih baik. (PostgreSQL)

C. Strategi *Partitioning*

Partisi paling umum untuk *Data Warehouse* adalah berdasarkan waktu.

Contoh:

```
fact_sales_2024
fact_sales_2025
fact_sales_2026
```

Atau lebih detail:

```
fact_sales_2026_01
fact_sales_2026_02
fact_sales_2026_03
```

Gunakan partisi jika:

- tabel fakta sangat besar;
- *query* sering memakai filter waktu;
- data historis perlu diarsipkan;
- proses load berjalan per periode;
- backup dan retensi perlu dipisah;
- data lama jarang diakses.

Contoh skema partisi PostgreSQL:

```
CREATE TABLE dw.fact_sales (
    sales_fact_key BIGINT,
    sales_date DATE NOT NULL,
    date_key INTEGER NOT NULL,
    customer_key INTEGER NOT NULL,
    product_key INTEGER NOT NULL,
    branch_key INTEGER NOT NULL,
    quantity NUMERIC(18,2),
    net_amount NUMERIC(18,2),
    margin_amount NUMERIC(18,2)
) PARTITION BY RANGE (sales_date);

Contoh partisi bulanan:
CREATE TABLE dw.fact_sales_2026_01
PARTITION OF dw.fact_sales
FOR VALUES FROM ('2026-01-01') TO ('2026-02-01');

CREATE TABLE dw.fact_sales_2026_02
PARTITION OF dw.fact_sales
FOR VALUES FROM ('2026-02-01') TO ('2026-03-01');
```

D. Risiko *Partitioning*

Partisi yang buruk dapat memperumit sistem.

Risiko:

- terlalu banyak partisi kecil;
- salah memilih kunci partisi;
- *query* tidak memakai filter partisi;
- maintenance lebih rumit;
- indeks per partisi perlu dikelola;
- statistik tidak diperbarui;
- load salah masuk partisi.

Referensi *Mastering Data Warehousing* menekankan bahwa partisi dapat mempercepat *query*, tetapi menambah kompleksitas pengelolaan partisi; strategi kompresi dan partisi juga harus ditinjau ulang sesuai perubahan volume data, kebutuhan *query*, dan frekuensi update.

Prinsip kritis: partisi harus mengikuti pola akses data. Jika pengguna selalu bertanya berdasarkan waktu, partisi waktu adalah pilihan alami.

10.3 *Baseline* dan Monitoring

Baseline adalah kondisi performa normal yang menjadi pembanding. Tanpa baseline, tim tidak tahu apakah sistem lambat karena masalah baru atau memang sejak awal lambat.

Monitoring harus mencatat:

- *query response time*;
- CPU utilization;
- memory usage;
- storage growth;
- I/O;
- *queue time*;
- *failed job*;
- ETL duration.

PostgreSQL documentation menyediakan fasilitas monitoring aktivitas database dan statistik untuk memantau aktivitas, sedangkan Snowflake documentation menyediakan pemantauan beban warehouse yang memperlihatkan rata-rata *query* berjalan atau mengantre dalam interval tertentu. (Dokumentasi Snowflake)

A. *Query Response Time*

Ukuran:

Jenis Query	Target Awal
dashboard ringkasan	< 5–10 detik
laporan bulanan	< 30 detik
ad hoc query ringan	< 60 detik
query analitik berat	sesuai SLA
ETL transformasi	sesuai jadwal batch

Yang harus dicatat:

- query ID;
- user;
- waktu mulai;
- durasi;
- tabel dibaca;
- baris diproses;
- status;
- error.

B. CPU Utilization

CPU tinggi bisa berarti:

- query kompleks;
- paralelisme tinggi;
- terlalu banyak user;
- ETL berat;
- indeks tidak optimal;
- agregasi terlalu besar.

Namun CPU tinggi tidak selalu buruk. Jika CPU tinggi dan *query* cepat, berarti sumber daya dipakai. Yang berbahaya adalah CPU tinggi, antrean panjang, dan respons lambat.

C. Memory Usage

Memori digunakan untuk:

- sort;
- hash join;
- agregasi;
- cache;
- temporary processing.

Gejala kekurangan memori:

- query lambat;
- banyak spill ke disk;
- temporary file besar;
- sort lambat;
- join berat.

D. Storage Growth

Pantau:

- data mentah;
- staging;
- tabel fakta;
- tabel dimensi;
- indeks;
- *materialized view*;
- backup;
- log;
- arsip.

Storage sering tumbuh bukan hanya karena data utama, tetapi karena:

- backup terlalu banyak;
- staging tidak dibersihkan;
- log tidak dibatasi;
- indeks berlebihan;
- *materialized view* tidak dikontrol;
- data historis tidak diarsipkan.

E. I/O

I/O tinggi menandakan banyak pembacaan atau penulisan storage.

Penyebab:

- full table scan;
- indeks tidak dipakai;
- query membaca banyak kolom;
- ETL load besar;
- backup;
- sort ke disk;
- partisi tidak efektif.

F. *Queue Time*

Queue time adalah waktu tunggu sebelum *query* diproses. Ini penting untuk melihat masalah *concurrency*.

Jika *queue time* tinggi:

- sumber daya kurang;
- banyak query bersamaan;
- query berat tidak dipisah;
- ETL bertabrakan dengan dashboard;
- workload perlu diprioritaskan.

Snowflake documentation menjelaskan bahwa *warehouse query load* mengukur rata-rata *query* yang berjalan atau mengantre, sehingga dapat digunakan untuk mengevaluasi performa warehouse dan kebutuhan concurrency. (Dokumentasi Snowflake)

G. *Failed Job*

Pantau semua job gagal:

- ETL gagal;
- refresh *materialized view* gagal;
- backup gagal;
- load partisi gagal;
- validasi data gagal;
- dashboard query gagal.

Metrik penting:

Metrik	Makna
jumlah job gagal	stabilitas proses
jenis error	akar masalah
durasi pemulihan	kesiapan operasi
frekuensi error	kualitas pipeline
dampak ke laporan	risiko bisnis

H. ETL Duration

ETL yang makin lama harus dianalisis.

Penyebab:

- volume data naik;
- indeks memperlambat load;
- transformasi tidak efisien;
- join ke dimensi lambat;
- data sumber lambat;
- staging penuh;
- partisi tidak optimal.

Contoh baseline:

Proses	**Baseline**	**Alert Jika**
extract_sales	15 menit	> 30 menit
transform_sales	25 menit	> 45 menit
load_fact_sales	20 menit	> 40 menit
refresh_dashboard_mart	10 menit	> 20 menit

10.4 Rumus Kapasitas Dasar

Capacity planning harus sederhana tetapi cukup akurat untuk estimasi awal. Tujuannya bukan menghasilkan angka sempurna, tetapi memberi dasar desain, anggaran, dan keputusan arsitektur.

A. Estimasi Ukuran *Fact Table*

Ukuran Fact = jumlah baris × ukuran rata-rata baris

Contoh:

Jumlah baris = 2.000.000 transaksi/bulan
Ukuran rata-rata baris = 500 byte

Ukuran per bulan = 2.000.000 × 500 byte
= 1.000.000.000 byte
≈ 1 GB/bulan

Catatan: angka ini hanya ukuran data mentah tabel fakta. Belum termasuk indeks, staging, backup, agregasi, dan overhead sistem.

B. Pertumbuhan Data

Data tahun ke-n = data awal × (1 + growth rate)^n

Untuk estimasi akumulatif beberapa tahun, hitung total per tahun lalu jumlahkan.

Contoh growth 20%:

Tahun	Data per Tahun
Tahun 1	12 GB
Tahun 2	14,4 GB
Tahun 3	17,28 GB
Tahun 4	20,736 GB
Tahun 5	24,883 GB

Total 5 tahun:

12 + 14,4 + 17,28 + 20,736 + 24,883 = 89,299 GB

C. Estimasi Storage Total

Storage Total = Raw Data + Staging + Warehouse + Index + Backup + Retention

Model praktis:

Warehouse Data = total fact + total dimension + aggregate

Index Overhead = Warehouse Data × persentase index

Backup = (Warehouse Data + Index) × jumlah salinan backup

Total Storage = Raw + Staging + Warehouse + Index + Backup + Log + Cadangan

Tambahkan cadangan kapasitas:

Cadangan Aman = 20%–30% dari total estimasi

D. Estimasi *Concurrency*

Kebutuhan Compute = jumlah user aktif × rata-rata query per user × kompleksitas query

Skor kompleksitas sederhana:

Kompleksitas	**Skor**
dashboard ringan	1
laporan standar	2
query dengan banyak join	3
agregasi besar	4
query eksploratif berat	5

Contoh:

30 user aktif × 2 query/user × skor kompleksitas 2 = 120 unit beban

Interpretasi:

- 0–50: ringan;
- 51–150: sedang;
- 151–300: berat;
- 300: perlu pemisahan workload / scaling.

Ini bukan ukuran teknis final, tetapi membantu diskusi kapasitas awal.

E. Estimasi Kebutuhan I/O dan ETL

Rumus sederhana:

> Data Diproses ETL = data harian masuk + data staging + data transformasi + data target

Jika data harian 50 GB dan proses menyentuh data tiga kali:

> I/O ETL minimum ≈ 50 GB × 3 = 150 GB per batch

Jika batch harus selesai dalam 2 jam:

> Throughput minimum = 150 GB / 2 jam = 75 GB/jam

Tambahkan overhead untuk validasi, indeks, log, dan backup.

10.5 Contoh Perhitungan

Kasus:

- 2 juta transaksi per bulan;
- ukuran rata-rata 1 baris = 500 byte;
- retensi 5 tahun;
- growth 20% per tahun;
- backup 2 salinan;
- index overhead 30%;
- staging = 20% dari warehouse aktif;
- raw archive = 100% data transaksi;
- cadangan kapasitas = 25%.

Langkah 1 — Hitung Data Bulanan Tahun Pertama

2.000.000 baris × 500 byte = 1.000.000.000 byte
≈ 1 GB per bulan

Data tahun pertama:

1 GB × 12 bulan = 12 GB

Langkah 2 — Hitung Pertumbuhan 5 Tahun

Tahun	Data Tahunan	Rumus
1	12,00 GB	12
2	14,40 GB	12 × 1,2
3	17,28 GB	14,4 × 1,2
4	20,74 GB	17,28 × 1,2
5	24,88 GB	20,74 × 1,2

Total data fakta 5 tahun:

12 + 14,40 + 17,28 + 20,74 + 24,88 = 89,30 GB

Langkah 3 — Tambahkan Dimensi dan Agregasi

Asumsi:

Dimensi + agregasi = 15% dari fact

Dimensi + agregasi = 89,30 × 15%
= 13,40 GB

Warehouse data:

Warehouse Data = 89,30 + 13,40
= 102,70 GB

Langkah 4 — Hitung Index Overhead

Index = Warehouse Data × 30%
= 102,70 × 0,30
= 30,81 GB

Langkah 5 — Hitung Raw dan Staging

Raw archive:
Raw = 89,30 GB

Staging:

Staging = Warehouse Data × 20%
= 102,70 × 0,20
= 20,54 GB

Langkah 6 — Hitung Backup

Backup 2 salinan untuk warehouse + index:

Backup = (Warehouse Data + Index) × 2
= (102,70 + 30,81) × 2
= 267,02 GB

Langkah 7 — Hitung Total Sebelum Cadangan

Total = Raw + Staging + Warehouse Data + Index + Backup

= 89,30 + 20,54 + 102,70 + 30,81 + 267,02

= 510,37 GB

Langkah 8 — Tambahkan Cadangan 25%

Cadangan = 510,37 × 25%
= 127,59 GB

Total akhir:

Total Storage = 510,37 + 127,59
= 637,96 GB

Rekomendasi kapasitas awal: minimal **650 GB**, lebih aman dibulatkan menjadi **1 TB** agar ada ruang untuk log, temporary file, pertumbuhan tak terduga, eksperimen, dan perubahan kebutuhan bisnis.

Ringkasan Perhitungan

Komponen	Estimasi
Fact 5 tahun	89,30 GB
Dimensi + agregasi	13,40 GB
Warehouse data	102,70 GB
Index 30%	30,81 GB
Raw archive	89,30 GB
Staging 20%	20,54 GB
Backup 2 salinan	267,02 GB
Total sebelum cadangan	510,37 GB
Cadangan 25%	127,59 GB
Total akhir	637,96 GB

Hands-on. Menghitung Kapasitas Storage dan Compute untuk DW 5 Tahun

Gunakan template berikut untuk latihan sertifikasi.

A. Input Kapasitas

Parameter	Nilai
transaksi per bulan	
ukuran rata-rata baris	
data per bulan	
retensi	
growth per tahun	
index overhead	
jumlah backup	
staging overhead	
raw archive	
cadangan kapasitas	

B. Rumus

Data Bulanan = jumlah transaksi per bulan × ukuran rata-rata baris

Data Tahunan = Data Bulanan × 12

Data Tahun ke-n = Data Tahun 1 × (1 + growth rate)^(n-1)

Total Fact Retensi = jumlah seluruh data tahunan selama periode retensi

Warehouse Data = Fact + Dimension + Aggregate

Index = Warehouse Data × Index Overhead

Backup = (Warehouse Data + Index) × jumlah salinan backup

Storage Total = Raw + Staging + Warehouse Data + Index + Backup + Cadangan

C. Template Perhitungan 5 Tahun

Tahun	Data Tahunan	Growth	Catatan
1			
2			
3			
4			
5			
Total			

D. Template Storage Total

Komponen	Rumus	Hasil
Fact total	total 5 tahun	
Dimensi + agregasi	fact × %	
Warehouse data	fact + dimensi + agregasi	
Index	warehouse × % index	
Raw archive	sesuai kebijakan	
Staging	warehouse × % staging	
Backup	warehouse + index × salinan	
Log dan temporary	estimasi	
Cadangan	total × % cadangan	
Total akhir	semua komponen	

E. Template Estimasi Compute

Parameter	Nilai
total user	
user aktif bersamaan	
query per user saat jam sibuk	
skor kompleksitas rata-rata	
workload ETL bersamaan	
target response time	
target ETL duration	

Rumus sederhana:

Beban Query = user aktif bersamaan × query per user × skor kompleksitas

Tambahkan skor ETL:

ETL	Skor
tidak bersamaan dengan jam kerja	0
ETL ringan saat jam kerja	25
ETL sedang saat jam kerja	50
ETL berat saat jam kerja	100

Total Beban Compute = Beban Query + Skor ETL

Interpretasi awal:

Total Beban	Rekomendasi Awal
0–50	compute kecil, dashboard ringan
51–150	compute sedang, perlu indeks dan agregasi
151–300	compute besar atau pisahkan workload
>300	perlu workload isolation, agregasi, partisi, dan capacity review

F. Contoh Estimasi Compute

Kasus:

User aktif bersamaan = 30
Query per user = 2
Skor kompleksitas = 2
ETL sedang saat jam kerja = 50

Hitung:

Beban Query = 30 × 2 × 2 = 120
Total Beban Compute = 120 + 50 = 170

Interpretasi:

- beban masuk kategori berat awal;
- perlu indeks pada kolom filter/join;
- perlu partisi fakta berdasarkan tanggal;
- dashboard sebaiknya membaca agregasi atau *materialized view*;
- ETL berat sebaiknya dipindah ke luar jam sibuk;
- workload analis dan dashboard sebaiknya dipisah.

G. Checklist Optimasi Performa

Area	Pertanyaan
Volume	apakah tabel fakta sudah dihitung pertumbuhannya?
Query	apakah query lambat sudah dianalisis dengan EXPLAIN?
Indeks	apakah indeks dibuat berdasarkan pola query nyata?
Partisi	apakah tabel besar di partisi berdasarkan waktu atau pola akses?
Agregasi	apakah dashboard membaca data agregat?
Materialized View	apakah query berat yang berulang sudah di prekomputasi?
Refresh	apakah frekuensi refresh sesuai kebutuhan bisnis?
Concurrency	apakah user aktif bersamaan sudah dihitung?
ETL	apakah ETL berjalan di luar jam sibuk?
Monitoring	apakah baseline durasi query dan ETL tersedia?
Storage	apakah raw, staging, index, backup, dan retensi dihitung?
Cost	apakah storage dan compute punya cadangan, tetapi tidak berlebihan?

Kesalahan Umum

Kesalahan yang sering terjadi:

- menghitung kapasitas hanya dari data utama;
- lupa menghitung indeks, staging, backup, log, dan agregasi;
- semua dashboard membaca tabel fakta detail;
- membuat indeks terlalu banyak;
- tidak memperbarui statistik database;
- tidak memakai partisi pada tabel historis besar;
- membuat partisi tetapi query tidak memakai filter partisi;
- ETL berjalan saat jam dashboard paling sibuk;
- tidak punya baseline performa;
- tidak memantau *queue time*;
- tidak membedakan workload analis dan dashboard;
- memilih compute besar untuk menutupi desain query yang buruk.

Prinsip kritis: menambah hardware atau compute tanpa memperbaiki model, query, indeks, dan partisi hanya akan membuat sistem buruk menjadi lebih mahal.

Ringkasan Bab

Performance tuning dan *capacity planning* memastikan *Data Warehouse* tetap cepat dan ekonomis ketika data, pengguna, dan kompleksitas bertambah. Performa dipengaruhi volume data, jumlah user, kompleksitas *query*, frekuensi *refresh*, indeks, partisi, agregasi, *materialized view*, dan *concurrency*. Kapasitas harus menghitung bukan hanya tabel fakta, tetapi juga raw data, staging, warehouse, indeks, backup, retensi, log, dan cadangan.

Poin penting:

- **Mulai optimasi dari pola query dan kebutuhan bisnis.**
- Indeks mempercepat pencarian, tetapi menambah storage dan memperlambat load.
- Partisi cocok untuk tabel besar, terutama data historis berbasis waktu.
- Agregasi dan *materialized view* mempercepat dashboard.
- *Concurrency* harus dihitung berdasarkan user aktif bersamaan, bukan total user.
- Monitoring membutuhkan baseline: waktu query, CPU, memory, storage, I/O, queue, job gagal, dan durasi ETL.
- Capacity planning harus memasukkan growth, backup, retensi, indeks, staging, dan cadangan.
- Compute besar bukan pengganti desain yang buruk.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Sebutkan faktor utama yang mempengaruhi performa *Data Warehouse*.
2. Mengapa indeks tidak boleh dibuat pada semua kolom?
3. Kapan tabel fakta sebaiknya di partisi?
4. Apa perbedaan agregasi biasa dan *materialized view*?
5. Mengapa *query response time* saja tidak cukup sebagai metrik monitoring?
6. Apa yang dimaksud *queue time* dan mengapa penting untuk *concurrency*?
7. Mengapa backup harus masuk perhitungan kapasitas?
8. Apa resiko menjalankan ETL berat pada jam pengguna aktif?

B. Studi Kasus

Sebuah *Data Warehouse* penjualan memiliki 2 juta transaksi per bulan, growth 20% per tahun, retensi 5 tahun, ukuran rata-rata baris 500 byte, backup 2 salinan, dan index overhead 30%.

Tugas:

- hitung data bulanan;
- hitung data tahunan;
- hitung total data 5 tahun;
- hitung index overhead;
- hitung backup;
- tentukan rekomendasi kapasitas storage;
- usulkan strategi indeks;
- usulkan strategi partisi;
- usulkan metrik monitoring.

C. Praktik

Buat dokumen *capacity planning* dengan format:

Komponen	Nilai
transaksi per bulan	
ukuran baris	
data per bulan	
retensi	
growth	
fact total	
dimensi + agregasi	
index overhead	
raw archive	
staging	
backup	
cadangan	
total storage	
user aktif bersamaan	
kompleksitas query	
rekomendasi compute	
strategi tuning	

BAB 11. Operasi Harian, Monitoring, Backup, dan *Disaster Recovery*

Data Warehouse tidak selesai ketika *go-live*. Justru setelah masuk *production*, tantangan sebenarnya dimulai: *ETL* harus berjalan tepat waktu, data harus selalu segar, error harus terdeteksi, storage tidak boleh penuh, *query* lambat harus dianalisis, akses user harus dikontrol, backup harus diuji, dan rencana pemulihan harus siap.

Sistem yang bagus di hari pertama bisa menjadi lambat, mahal, dan beresiko jika operasi hariannya lemah. Karena itu, operasi *Data Warehouse* harus memiliki disiplin: **cek harian, monitoring, backup, restore test, maintenance, dan *disaster recovery***.

Referensi *Mastering Data Warehousing* menekankan bahwa pengelolaan *Data Warehouse* mencakup pemantauan, optimasi performa, penanganan kegagalan, backup, recovery, refresh, archiving, security, access control, dan data quality checks agar sistem tetap dapat diandalkan untuk pengambilan keputusan. PostgreSQL juga menyediakan dokumentasi resmi untuk monitoring aktivitas database, statistik server, serta analisis *query* lambat menggunakan `EXPLAIN`. (PostgreSQL)

Golden Rule: *Data Warehouse production* yang sehat bukan yang tidak pernah gagal, tetapi yang **gagalnya cepat terdeteksi, cepat dipulihkan, dan tidak merusak kepercayaan data**.

11.1 Operasi Harian

Operasi harian adalah rutinitas minimum untuk menjaga *Data Warehouse* tetap sehat. Aktivitas ini sebaiknya dilakukan oleh tim operasi data, *data engineer*, DBA, atau tim platform.

Tujuan operasi harian:

- memastikan data terbaru sudah masuk;
- memastikan *ETL* berjalan;
- memastikan dashboard tidak menampilkan data basi;
- memastikan error ditangani;
- memastikan storage cukup;
- memastikan *query* lambat terdeteksi;
- memastikan akses user tidak menyimpang;
- memastikan backup berhasil.

Operasi harian harus berbasis checklist. Jangan mengandalkan ingatan.

A. Cek *ETL*

ETL adalah jantung harian *Data Warehouse*. Jika *ETL* gagal, dashboard bisa menampilkan data lama atau data tidak lengkap.

Yang dicek:

- apakah job *ETL* berjalan sesuai jadwal;
- apakah semua tahap selesai;
- berapa durasi proses;
- apakah ada job gagal;
- apakah ada job yang berjalan terlalu lama;
- apakah ada batch yang hanya sukses sebagian;
- apakah proses bisa diulang jika gagal.

Contoh status:

Proses	**Jadwal**	**Status**	**Durasi**	**Catatan**
extract_sales	01:00	sukses	15 menit	normal
transform_sales	01:20	sukses	25 menit	normal
load_fact_sales	01:50	gagal	-	kode produk tidak valid
refresh_mart_sales	02:30	belum jalan	-	menunggu fact

Prinsip kritis: jangan hanya cek “job selesai”. Cek juga **jumlah baris, error, dan rekonsiliasi**.

B. Cek *Data Refresh*

Data refresh memastikan data di *warehouse* sesuai periode yang seharusnya.

Yang dicek:

- tanggal data terakhir;
- jumlah data harian;
- keterlambatan sumber;
- apakah data sudah masuk ke *data mart*;
- apakah dashboard membaca data terbaru;
- apakah ada periode kosong.

Contoh query:

```
SELECT MAX(sales_date) AS latest_sales_date
FROM dw.fact_sales;
```

Contoh indikator:

Dataset	Target Refresh	Data Terakhir	Status
Penjualan	H-1 sebelum 07:00	H-1	OK
Pelanggan	harian	H-1	OK
Produk	mingguan	minggu ini	OK
Proyek	mingguan	minggu lalu	terlambat
QHSE	harian	H-2	terlambat

Jika data terlambat, pengguna harus diberi tahu. Dashboard yang menampilkan data basi tanpa peringatan dapat menyesatkan keputusan.

C. Cek *Error Log*

Error log membantu menemukan masalah kualitas data dan masalah proses.

Yang dicek:

- jumlah error;
- jenis error;
- sumber error;
- apakah error berulang;
- apakah error mempengaruhi KPI;
- apakah error perlu eskalasi ke *data owner*.

Contoh error:

Jenis Error	Penyebab	Tindakan
null value	field wajib kosong	kirim ke data steward
invalid branch code	kode cabang tidak ada	cek master cabang
duplicate invoice	file dikirim ulang	deduplikasi
API timeout	sumber tidak responsif	retry dan eskalasi
load failed	constraint gagal	rollback batch

Prinsip kritis: error kecil yang diabaikan setiap hari akan menjadi masalah besar saat audit.

D. Cek Storage

Storage harus dipantau karena *Data Warehouse* menyimpan data historis, staging, log, indeks, agregasi, dan backup.

Yang dicek:

- kapasitas total;
- kapasitas terpakai;
- pertumbuhan harian / mingguan;
- ukuran tabel fakta;
- ukuran indeks;
- ukuran staging;
- ukuran log;
- ukuran backup;
- ruang temporary.

Contoh indikator:

Area	Ambang Peringatan
storage terpakai	> 80%
growth mingguan	> baseline
staging tidak dibersihkan	> 7 hari
log terlalu besar	> retensi
backup gagal	harus segera ditangani

Storage penuh dapat menyebabkan *ETL* gagal, backup gagal, dashboard lambat, dan risiko korupsi operasional.

E. Cek *Query* Lambat

Query lambat harus dipantau karena berdampak langsung ke pengguna.

Yang dicek:

- query dengan durasi paling lama;
- query yang paling sering dijalankan;
- query yang membaca tabel besar;
- query dashboard yang gagal;
- query analis yang membebani sistem;
- query yang tidak memakai indeks atau partisi.

PostgreSQL menjelaskan bahwa monitoring aktivitas database dapat dilakukan melalui statistik internal dan alat sistem seperti `ps`, `top`, `iostat`, `vmstat`; setelah query bermasalah ditemukan, investigasi dapat dilanjutkan dengan `EXPLAIN`. (PostgreSQL)

Contoh query analisis sederhana:

```
EXPLAIN ANALYZE
SELECT
    product_key,
    SUM(net_amount)
FROM dw.fact_sales
WHERE sales_date >= DATE '2026-01-01'
GROUP BY product_key;
```

F. Cek Akses User

Akses user harus dikontrol secara berkala. Risiko umum adalah *access creep*: hak akses terus bertambah, tetapi jarang dicabut.

Yang dicek:

- user baru;
- user keluar / pindah unit;
- role yang berubah;
- akses data sensitif;
- akses admin;
- aktivitas ekspor;
- login gagal;
- akses di luar jam normal;
- akses yang tidak sesuai fungsi.

Checklist akses:

Pemeriksaan	Tujuan
user aktif	memastikan hanya user sah
role	memastikan sesuai fungsi
akses sensitif	mencegah kebocoran
ekspor data	mencegah penyalahgunaan
admin privilege	mencegah akses berlebihan
audit log	membuktikan aktivitas

G. Cek Backup

Backup harus dicek setiap hari atau sesuai SLA. Jangan menunggu insiden untuk mengetahui backup gagal.

Yang dicek:

- backup berjalan;
- ukuran backup wajar;
- backup tersimpan di lokasi benar;
- log backup sukses;
- backup tidak korup;
- retensi backup sesuai kebijakan;
- backup penting sudah terenkripsi;
- restore test dijadwalkan.

PostgreSQL menyebut tiga pendekatan backup utama: SQL dump, file system level backup, dan continuous archiving. (PostgreSQL) Untuk pemulihan berbasis waktu, continuous archiving membutuhkan urutan arsip WAL yang lengkap sejak awal backup agar recovery berhasil. (PostgreSQL)

Prinsip kritis: backup sukses belum cukup. Yang menentukan adalah **restore berhasil**.

11.2 Backup

Backup adalah mekanisme perlindungan data. Dalam *Data Warehouse*, backup tidak hanya mencakup database utama. Backup juga harus mencakup metadata, script *ETL*, konfigurasi dashboard, log penting, dan dokumentasi operasional.

Jenis backup:

- *full backup*;
- *incremental backup*;
- snapshot;
- archive;
- backup metadata;
- backup script *ETL*.

A. *Full Backup*

Full backup menyalin seluruh database atau seluruh dataset penting.

Kelebihan:

- mudah dipahami;
- restore lebih sederhana;
- cocok untuk baseline backup;
- baik untuk backup periodik.

Kekurangan:

- membutuhkan storage besar;
- waktu backup lama;
- membebani sistem;
- tidak efisien jika data besar.

Cocok untuk:

- backup mingguan;
- backup sebelum deployment besar;
- backup sebelum migrasi;
- backup sebelum perubahan struktur besar.

B. *Incremental Backup*

Incremental backup hanya menyimpan perubahan sejak backup terakhir.

Kelebihan:

- lebih cepat;
- hemat storage;
- cocok untuk database besar;
- baik untuk backup harian.

Kekurangan:

- restore lebih kompleks;
- bergantung pada rangkaian backup sebelumnya;
- perlu manajemen retensi yang rapi;
- perlu diuji secara berkala.

Gunakan incremental backup bersama dokumentasi restore yang jelas.

C. Snapshot

Snapshot menangkap kondisi sistem pada titik waktu tertentu.

Cocok untuk:

- sebelum deployment;
- sebelum perubahan skema;
- sebelum load historis besar;
- sebelum eksperimen transformasi;
- pemulihan cepat dari perubahan salah.

Snapshot berguna, tetapi jangan menggantikan backup formal. Snapshot bisa bergantung pada platform storage dan tidak selalu cukup untuk pemulihan jangka panjang.

D. Archive

Archive adalah penyimpanan data lama yang jarang diakses tetapi masih diperlukan.

Contoh:

- raw file historis;
- data transaksi lama;
- log ETL lama;
- audit trail;
- backup tahunan;
- data periode audit.

Archive membantu:

- mengurangi ukuran data aktif;
- mempercepat query;
- menurunkan beban storage aktif;
- memenuhi retensi audit;
- mendukung *disaster recovery*.

Aturan archive harus jelas:

Data	**Retensi Aktif**	**Archive**	**Catatan**
transaksi	2 tahun	5–10 tahun	sesuai kebijakan
log ETL	1 tahun	3 tahun	untuk audit
staging mentah	3 bulan	jika diperlukan	tergantung risiko
backup bulanan	1 tahun	3–5 tahun	sesuai compliance

E. Backup Metadata

Metadata sering dilupakan. Padahal metadata menjelaskan arti data, asal data, transformasi, lineage, dan definisi KPI.

Yang harus di backup:

- *data dictionary*;
- *ETL mapping*;
- *data lineage*;
- definisi KPI;
- skema database;
- access matrix;
- konfigurasi *data mart*;
- konfigurasi dashboard;
- dokumentasi UAT.

Referensi yang digunakan menekankan bahwa metadata mendukung pemahaman data, *governance*, compliance, data quality, dokumentasi, integrasi sistem, dan visibilitas ownership serta access restrictions.

F. Backup Script *ETL*

Script *ETL* adalah bagian dari sistem produksi. Jika script hilang, data warehouse mungkin tidak bisa diperbarui.

Yang harus di backup:

- script extract;
- script transform;
- script load;
- konfigurasi koneksi;
- jadwal job;
- file parameter;
- definisi environment;
- dependency;
- versi script;
- dokumentasi error handling.

Prinsip kritis: backup database tanpa backup script *ETL* hanya memulihkan data lama, bukan memulihkan kemampuan sistem untuk terus hidup.

11.3 *Disaster Recovery*

Disaster Recovery atau DR adalah rencana untuk memulihkan sistem setelah gangguan besar. Gangguan bisa berupa kegagalan server, kerusakan storage, kesalahan manusia, serangan siber, kebakaran, bencana alam, atau kegagalan data center.

Komponen utama:

- RPO;
- RTO;
- *secondary site*;
- *restore test*;
- *failover drill*.

A. RPO

Recovery Point Objective atau RPO adalah batas maksimum kehilangan data yang dapat diterima.

Contoh:

Sistem	RPO
dashboard strategis bulanan	24 jam
transaksi harian	4 jam
monitoring operasional	15 menit
audit log	serendah mungkin

Jika RPO = 24 jam, organisasi menerima kemungkinan kehilangan data maksimal 24 jam. Jika RPO = 15 menit, sistem backup dan replikasi harus jauh lebih kuat.

Prinsip kritis: RPO rendah berarti biaya dan kompleksitas lebih tinggi.

B. RTO

Recovery Time Objective atau RTO adalah target waktu maksimal untuk memulihkan sistem.

Contoh:

Sistem	RTO
dashboard direksi	4 jam
laporan bulanan	1 hari
data mart operasional	2 jam
audit repository	8 jam

RTO menjawab: “berapa lama organisasi masih bisa beroperasi tanpa sistem ini?”

RPO dan RTO harus disetujui oleh bisnis, bukan hanya teknis.

C. *Secondary Site*

Secondary site adalah lokasi cadangan untuk pemulihan.

Bentuknya bisa:

- server cadangan;
- database replika;
- storage cadangan;
- lokasi berbeda;
- lingkungan DR yang siap diaktifkan;
- backup di lokasi terpisah.

Tujuan:

- menghindari satu titik gagal;
- mempercepat pemulihan;
- melindungi dari bencana lokasi;
- mendukung *business continuity*.

Desain *secondary site* harus memperhatikan:

- sinkronisasi data;
- keamanan;
- biaya;
- jarak lokasi;
- bandwidth;
- uji pemulihan;
- prosedur aktivasi.

D. *Restore Test*

Restore test adalah uji pemulihan dari backup. Ini wajib dilakukan berkala.

Yang diuji:

- apakah file backup bisa dibaca;
- apakah database bisa dipulihkan;
- apakah skema lengkap;
- apakah data valid;
- apakah *ETL* bisa berjalan kembali;
- apakah dashboard bisa membaca data;
- apakah akses user kembali benar;
- berapa lama restore berlangsung.

Contoh jadwal:

Jenis Test	Frekuensi
restore kecil	bulanan
restore penuh	kuartalan
restore metadata	kuartalan
restore ETL	setiap release besar
simulasi DR	1–2 kali per tahun

E. Failover Drill

Failover drill adalah latihan memindahkan operasi ke sistem cadangan.

Tujuan:

- menguji kesiapan tim;
- menguji prosedur;
- mengukur RTO;
- menguji *secondary site*;
- menemukan gap sebelum insiden nyata;
- melatih komunikasi darurat.

Langkah sederhana:

1. Umumkan drill ke tim terkait.
2. Bekukan perubahan non-kritis.
3. Simulasikan kegagalan production.
4. Aktifkan secondary site.
5. Validasi data dan dashboard.
6. Catat waktu pemulihan.
7. Kembalikan operasi normal.
8. Buat laporan hasil drill.

Prinsip kritis: DR plan yang tidak pernah diuji hanyalah dokumen optimis.

11.4 *Maintenance*

Maintenance menjaga *Data Warehouse* tetap cepat, bersih, dan stabil. Maintenance harus dijadwalkan, bukan dilakukan hanya saat sistem bermasalah.

Aktivitas utama:

- `VACUUM` / `ANALYZE`;
- rebuild index;
- update statistics;
- partition maintenance;
- cleanup staging;
- archive old data.

A. `VACUUM` / `ANALYZE`

Pada PostgreSQL, `VACUUM` digunakan untuk membersihkan ruang dari tuple lama dan menjaga kesehatan database. Dokumentasi PostgreSQL menyebut bahwa database membutuhkan maintenance berkala bernama vacuuming, dan untuk banyak instalasi autovacuum sudah cukup, walau parameter dapat disesuaikan. (PostgreSQL) Dokumentasi `VACUUM` juga menjelaskan bahwa `VACUUM ANALYZE` menjalankan `VACUUM` lalu `ANALYZE`, sehingga berguna untuk script maintenance rutin. (PostgreSQL)

Contoh:

```
VACUUM ANALYZE dw.fact_sales;
```

Gunakan saat:

- setelah load besar;
- setelah delete / update besar;
- setelah arsip data;
- ketika query mulai lambat;
- sesuai jadwal maintenance.

B. Rebuild Index

Indeks dapat mengalami bloat atau tidak lagi optimal setelah banyak perubahan data. Rebuild index membantu memperbaiki struktur indeks.

Contoh aktivitas:

```
REINDEX INDEX idx_fact_sales_date;
```

Atau pada tabel tertentu:

```
REINDEX TABLE dw.fact_sales;
```

Gunakan hati-hati di *production*. Jadwalkan di luar jam sibuk karena rebuild index dapat membebani sistem.

C. *Update Statistics*

Statistik membantu optimizer memilih rencana *query* terbaik. Jika statistik lama, query bisa memilih strategi eksekusi yang buruk.

PostgreSQL memiliki perintah `ANALYZE` untuk mengumpulkan statistik isi tabel, sehingga perencana query dapat mengambil keputusan lebih baik. (PostgreSQL)

Contoh:

```
ANALYZE dw.fact_sales;
ANALYZE dw.dim_customer;
```

Jalankan setelah:

- load data besar;
- perubahan distribusi data;
- penambahan partisi;
- update besar;
- performa query berubah.

D. *Partition Maintenance*

Untuk tabel fakta besar, partisi harus dirawat.

Kegiatan:

- membuat partisi baru sebelum periode berjalan;
- memastikan data masuk partisi benar;
- mengarsipkan partisi lama;
- membuat indeks pada partisi baru;
- menghapus partisi yang melewati retensi;
- update statistik partisi;
- memvalidasi query memakai partisi.

Contoh jadwal:

Waktu	Aktivitas
akhir bulan	buat partisi bulan berikutnya
awal bulan	validasi partisi bulan lalu
kuartalan	arsip partisi lama
tahunan	review strategi partisi

E. *Cleanup Staging*

Staging sering menjadi sumber pemborosan storage.

Yang dibersihkan:

- file raw yang sudah diproses;
- tabel staging lama;
- data batch gagal yang sudah diselesaikan;
- file sementara;
- log debug yang melewati retensi;
- duplicate file.

Namun jangan menghapus staging tanpa aturan. Staging mungkin dibutuhkan untuk audit dan troubleshooting.

Contoh retensi:

Data Staging	Retensi
raw harian	30–90 hari
staging clean	30 hari
error rows	90–180 hari
log ETL	1 tahun
file mentah audit	sesuai kebijakan

F. *Archive Old Data*

Data lama tidak harus selalu aktif. Jika tetap disimpan di tabel utama, query bisa melambat dan storage membengkak.

Strategi archive:

- pisahkan data aktif dan historis;
- archive berdasarkan periode;
- simpan ringkasan agregat;
- simpan raw file jika diperlukan;
- dokumentasikan lokasi archive;
- pastikan archive bisa dipulihkan;
- pastikan retensi sesuai kebijakan.

Contoh:

```
Data aktif: 2 tahun terakhir
Data archive: tahun ke-3 sampai ke-5
Data agregat: 10 tahun
```

Prinsip kritis: archive bukan tempat membuang data. Archive adalah bagian dari strategi retensi dan pemulihan.

Hands-on. Template Checklist Operasi Harian *Data Warehouse*

Gunakan template ini untuk operasi harian tim *Data Warehouse*. Checklist dapat ditempatkan di dokumen kerja, sistem tiket, atau dashboard operasi.

A. Identitas Operasi

Elemen	Isi
Tanggal operasi	
Petugas	
Environment	Production
Sistem / domain	
Jam pengecekan	
Status umum	Normal / Warning / Critical

B. Checklist *ETL*

Item	Status	Catatan
Semua job ETL selesai		
Tidak ada job gagal		
Durasi ETL sesuai baseline		
Row count sesuai ekspektasi		
Error count dalam batas wajar		
Rekonsiliasi sukses		
Job berikutnya terjadwal		

C. Checklist *Data Refresh*

Dataset	Target Refresh	Data Terakhir	Status	Catatan
FactSales	H-1 sebelum 07:00			
DimCustomer	harian			
DimProduct	mingguan			
DimBranch	bulanan / saat berubah			
Data mart penjualan	harian			

D. Checklist *Error Log*

Jenis Error	Jumlah	Dampak	Tindakan
null field			
duplicate			
invalid reference			
load failure			
API / source failure			

E. Checklist Storage

Area	Terpakai	Ambang	Status	Catatan
Database utama		80%		
Staging		sesuai retensi		
Log		sesuai retensi		
Backup		sesuai kapasitas		
Temporary		sesuai baseline		

F. Checklist *Query* dan Performa

Item	Status	Catatan
Tidak ada query sangat lambat		
Dashboard utama normal		
CPU dalam batas normal		
Memory dalam batas normal		
I/O dalam batas normal		
Tidak ada antrean query panjang		
ETL tidak mengganggu dashboard		

G. Checklist Akses User

Item	Status	Catatan
Tidak ada login mencurigakan		
Tidak ada akses sensitif tanpa otorisasi		
Tidak ada ekspor data besar tanpa alasan		
User keluar sudah dicabut		
Perubahan role terdokumentasi		
Akses admin di review		

H. Checklist Backup

Item	Status	Catatan
Backup terakhir sukses		
Ukuran backup wajar		
Backup tersimpan di lokasi benar		
Backup terenkripsi jika sensitif		
Retensi backup sesuai kebijakan		
Restore test berikutnya terjadwal		

I. Checklist Maintenance

Item	Status	Catatan
VACUUM / ANALYZE sesuai jadwal		
Statistik tabel utama baru		
Partisi bulan berikutnya tersedia		
Staging lama dibersihkan		
Log lama diarsipkan		
Data lama sesuai retensi		

J. Eskalasi

Kondisi	Eskalasi Ke	SLA Respons
ETL gagal	Data Engineer	1 jam
Data refresh terlambat	Data Owner + Data Engineer	2 jam
Backup gagal	DBA / Platform Admin	1 jam
Storage > 85%	Platform Admin	4 jam
Akses mencurigakan	Security Officer	segera
Dashboard direksi gagal	Tim Data + Sponsor	segera
Data KPI salah	Data Owner + Auditor jika perlu	segera

K. Catatan Harian

Tanggal:
Petugas:
Status umum:
Masalah utama:
Tindakan:
Eskalasi:
Rencana tindak lanjut:

Checklist Mingguan

Selain operasi harian, lakukan review mingguan:

Area	Pertanyaan
ETL	apakah durasi meningkat?
Data quality	apakah error berulang?
Storage	apakah growth sesuai prediksi?
Query	query apa yang paling lambat?
User access	apakah ada akses tidak perlu?
Backup	apakah backup mingguan sukses?
Maintenance	apakah vacuum/analyze berjalan?
Incident	apakah ada insiden yang perlu RCA?

Checklist Bulanan

Lakukan review bulanan:

Area	Aktivitas
Capacity	review growth storage dan compute
Access	review role dan akses sensitif
Backup	restore test sampel
DR	cek kesiapan prosedur
Documentation	update data dictionary dan lineage
Performance	review indeks dan partisi
Cost	review biaya storage, compute, backup
Governance	laporkan isu kualitas data ke data owner

Kesalahan Umum Operasi

Kesalahan yang sering terjadi:

- hanya mengecek dashboard, bukan *ETL*;
- tidak memeriksa data refresh;
- error log dibiarkan menumpuk;
- backup tidak pernah diuji restore;
- staging tidak dibersihkan;
- partisi baru lupa dibuat;
- statistik tidak diperbarui;
- query lambat tidak dianalisis;
- akses user lama tidak dicabut;
- data lama tidak diarsipkan;
- DR plan tidak pernah diuji;
- tidak ada checklist operasi harian;
- insiden diperbaiki tetapi tidak dibuat *root cause analysis*.

Prinsip kritis: operasi yang buruk akan menghancurkan *Data Warehouse* yang desainnya bagus.

Ringkasan Bab

Operasi harian menjaga *Data Warehouse* tetap hidup setelah *production*. Aktivitas utama meliputi cek *ETL*, cek data refresh, cek error log, cek storage, cek *query* lambat, cek akses user, dan cek backup. Backup harus mencakup full backup, incremental backup, snapshot, archive, metadata, dan script *ETL*. *Disaster Recovery* harus menetapkan RPO, RTO, secondary site, restore test, dan failover drill. Maintenance meliputi `VACUUM / ANALYZE`, rebuild index, update statistics, partition maintenance, cleanup staging, dan archive old data.

Poin penting:

- **Production DW harus dipantau setiap hari.**
- Data yang terlambat harus diberi tanda, bukan disembunyikan.
- Error log adalah sinyal kualitas data dan stabilitas proses.
- Backup tanpa restore test belum dapat dipercaya.
- RPO dan RTO harus disetujui bisnis.
- DR plan wajib diuji melalui *failover drill*.
- Maintenance harus dijadwalkan, bukan dilakukan saat panik.
- Archive mengurangi beban data aktif dan mendukung retensi.
- Checklist harian, mingguan, dan bulanan membuat operasi lebih disiplin.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Mengapa operasi *Data Warehouse* tidak selesai setelah *go-live*?
2. Apa saja yang harus dicek dalam operasi harian *ETL*?
3. Mengapa *data refresh* perlu dipantau secara khusus?
4. Apa perbedaan *full backup*, *incremental backup*, snapshot, dan archive?
5. Mengapa metadata dan script *ETL* harus ikut di backup?
6. Jelaskan perbedaan RPO dan RTO.
7. Mengapa *restore test* lebih penting daripada sekadar status backup sukses?
8. Apa fungsi `VACUUM`, `ANALYZE`, dan rebuild index dalam maintenance?

B. Studi Kasus

Sebuah *Data Warehouse* penjualan nasional sudah berjalan di *production*. Suatu pagi, dashboard direksi masih menampilkan data dua hari lalu. Setelah dicek, *ETL* gagal karena kode cabang baru belum masuk ke dimensi cabang.

Tugas:

- identifikasi masalah operasi;
- tentukan langkah pengecekan;
- tentukan siapa yang harus eskalasi;
- buat tindakan pemulihan;
- buat tindakan pencegahan agar tidak berulang;
- tentukan perubahan checklist harian.

C. Praktik

Buat checklist operasi harian untuk *Data Warehouse* Anda sendiri dengan format:

Area	Pemeriksaan	Status	Catatan	Tindak Lanjut
ETL				
Data Refresh				
Error Log				
Storage				
Query Lambat				
Akses User				
Backup				
Maintenance				

BAGIAN IV. SDM, BIAYA, SUSTAINABILITY, DAN SERTIFIKASI

BAB 12. Perencanaan SDM Data Warehouse

Data Warehouse bukan hanya proyek teknologi. Ia adalah proyek organisasi. Di dalamnya ada kebutuhan bisnis, arsitektur data, integrasi, kualitas data, keamanan, audit, dashboard, biaya, operasi, dan perubahan budaya kerja. Karena itu, keberhasilan *Data Warehouse* sangat ditentukan oleh **SDM yang tepat**.

Banyak proyek *Data Warehouse* gagal bukan karena database tidak mampu, tetapi karena tim tidak lengkap: tidak ada *data owner*, tidak ada *data steward*, tidak ada arsitek data, tidak ada pengelola kualitas data, tidak ada penguji, tidak ada pemilik KPI, dan tidak ada orang yang bertanggung jawab setelah sistem masuk produksi.

Referensi implementasi *Data Warehouse* menekankan bahwa proyek harus dimulai dengan *project charter*, pembentukan tim, identifikasi *stakeholder*, *requirement gathering*, *data modeling*, *ETL*, QA, *performance tuning*, UAT, deployment, dan pemeliharaan. Ini menunjukkan bahwa implementasi DW adalah kerja lintas peran, bukan hanya pekerjaan teknis satu orang. SFIA juga menempatkan *data management* sebagai kemampuan untuk mengembangkan dan menjalankan rencana, kebijakan, dan praktik yang mengontrol, melindungi, serta mengoptimalkan nilai dan tata kelola aset data. (SFIA)

Golden Rule: tim *Data Warehouse* yang baik bukan yang paling besar, tetapi yang **jelas perannya, jelas tanggung jawabnya, jelas kompetensinya, dan jelas akuntabilitas datanya**.

12.1 Struktur Tim Minimal

Tim minimal *Data Warehouse* harus mencakup peran bisnis, teknis, tata kelola, keamanan, dan manajemen proyek. Satu orang boleh merangkap beberapa peran pada organisasi kecil, tetapi **fungsi kritisnya tetap harus ada**.

Struktur minimal:

- Sponsor bisnis.
- *Product owner data*.
- *Data architect*.
- *Data engineer*.
- *ETL developer*.
- DBA / *platform engineer*.
- *BI developer*.
- *Data analyst*.
- *Data steward*.
- *Security / compliance officer*.
- *Project manager*.

A. Sponsor Bisnis

Sponsor bisnis adalah pihak eksekutif yang memberi mandat, arah, prioritas, dan dukungan organisasi.

Tanggung jawab:

- menetapkan tujuan strategis;
- menyetujui *scope*;
- mengamankan anggaran;
- menyelesaikan konflik antar unit;
- memastikan hasil DW dipakai untuk keputusan;
- memberi dukungan saat terjadi resistensi.

Tanpa sponsor bisnis, proyek DW sering berhenti di tengah jalan karena dianggap hanya proyek teknis.

Prinsip kritis: sponsor bisnis harus peduli pada keputusan, KPI, dan nilai organisasi, bukan hanya pada status proyek.

B. *Product Owner Data*

Product owner data adalah pemilik kebutuhan produk data. Perannya menjembatani kebutuhan bisnis dengan tim teknis.

Tanggung jawab:

- menentukan prioritas fitur DW;
- menyusun daftar kebutuhan laporan;
- mengelola backlog;
- menyetujui definisi KPI bersama *data owner*;
- memastikan dashboard menjawab kebutuhan pengguna;
- memutuskan prioritas MVP, pilot, dan produksi.

Contoh keputusan:

- domain data mana dulu;
- KPI apa yang masuk fase pertama;
- dashboard mana yang paling penting;
- laporan lama mana yang diganti;
- data mana yang belum layak masuk produksi.

C. *Data Architect*

Data architect merancang arsitektur DW secara menyeluruh.

Tanggung jawab:

- mendesain arsitektur data;
- menentukan pola *staging*, DW core, dan *data mart*;
- mendesain *dimensional modeling*;
- menentukan *grain*;
- memilih *star schema* atau *snowflake schema*;
- menetapkan standar tabel dan kolom;
- merancang metadata dan *data lineage*;
- memastikan desain dapat tumbuh.

Referensi desain DW menekankan bahwa *logical design* menerjemahkan kebutuhan bisnis ke model konseptual, sedangkan *physical design* menerjemahkan model logis menjadi tabel, kolom, tipe data, *constraint*, *primary key*, *foreign key*, dan relasi fisik.

Prinsip kritis: arsitek data tidak hanya menggambar diagram. Ia menentukan apakah DW mudah dikembangkan atau menjadi beban jangka panjang.

D. *Data Engineer*

Data engineer membangun aliran data dari sumber ke DW.

Tanggung jawab:

- membangun *pipeline ETL/ELT*;
- mengelola *staging*;
- membuat proses *extract*;
- membuat transformasi data;
- membuat *load* ke dimensi dan fakta;
- membuat log dan error handling;
- membuat rekonsiliasi;
- menjaga pipeline stabil.

SFIA menempatkan *data engineering* sebagai kemampuan yang terkait dengan *data management*, *data modelling and design*, *database design*, *database administration*, dan *data visualisation*. (SFIA)

Catatan kritis: peran *data engineer* sebaiknya tidak terlalu sering di rangkap, karena ini adalah peran berat dan sangat menentukan stabilitas DW.

E. *ETL Developer*

ETL developer fokus pada detail proses *extract*, *transform*, dan *load*. Pada tim kecil, peran ini sering dirangkap oleh *data engineer*.

Tanggung jawab:

- membuat script *extract*;
- membuat aturan transformasi;
- menerapkan *cleansing*;
- menerapkan deduplikasi;
- membuat *source-to-target mapping*;
- membuat proses *incremental load*;
- membuat tabel error;
- membuat log proses.

Referensi ETL menjelaskan bahwa proses ETL mengambil data dari berbagai sumber, mentransformasikannya menjadi format konsisten, lalu memuatnya ke DW atau *data mart*.

F. DBA / *Platform Engineer*

DBA atau *platform engineer* menjaga platform database, performa, backup, keamanan teknis, dan operasi.

Tanggung jawab:

- instalasi dan konfigurasi database;
- pengelolaan storage;
- backup dan restore;
- monitoring performa;
- tuning indeks dan partisi;
- pengelolaan user dan role teknis;
- pemeliharaan database;
- dukungan *disaster recovery*.

SFIA mendefinisikan *database administration* sebagai kegiatan memasang, mengkonfigurasi, memonitor, memelihara database dan *data store*, memastikan performa dan keamanan, serta menyesuaikan dengan perkembangan teknologi. (SFIA)

G. *BI Developer*

BI developer membuat laporan, *dashboard*, dan visualisasi.

Tanggung jawab:

- menerjemahkan KPI menjadi dashboard;
- membuat visualisasi;
- membuat filter;
- mengatur akses dashboard;
- membuat semantic layer sederhana;
- menguji konsistensi angka;
- bekerja dengan pengguna bisnis;
- memastikan dashboard mudah dipahami.

Referensi arsitektur DW menempatkan *presentation layer* sebagai lapisan yang menyediakan laporan, dashboard, visualisasi, dan akses data bagi pengguna akhir.

Prinsip kritis: dashboard yang indah tetapi angkanya tidak tervalidasi adalah risiko, bukan prestasi.

H. *Data Analyst*

Data analyst menggunakan DW untuk menjawab pertanyaan bisnis.

Tanggung jawab:

- melakukan eksplorasi data;
- membuat analisis trend;
- memvalidasi KPI;
- membantu UAT;
- memberi masukan ke model data;
- menemukan anomali;
- membantu pengguna bisnis memahami data.

Analis sering menjadi pengguna awal yang menemukan apakah DW benar-benar berguna atau hanya rapi secara teknis.

I. *Data Steward*

Data steward menjaga kualitas, definisi, dan disiplin data harian.

Tanggung jawab:

- menjaga *data dictionary*;
- memastikan definisi KPI konsisten;
- memantau kualitas data;
- menindaklanjuti data error;
- mengelola isu data lintas unit;
- bekerja dengan *data owner*;
- menjaga metadata dan aturan bisnis.

Referensi metadata menekankan bahwa metadata membantu *data governance*, compliance, *data quality management*, dokumentasi, sistem integrasi, serta visibilitas ownership, akses, dan pola penggunaan data.

J. *Security / Compliance Officer*

Peran ini memastikan DW aman, legal, dan siap audit.

Tanggung jawab:

- klasifikasi data;
- kontrol akses;
- masking;
- audit log;
- review akses;
- compliance;
- pengelolaan risiko data sensitif;
- review *data sharing*;
- review retensi data.

DAMA-DMBOK membantu organisasi menata, mengelola, melindungi, mengintegrasikan, dan mengoptimalkan aset data agar selaras dengan strategi bisnis, compliance, dan perkembangan teknologi. (DAMA International®)

K. *Project Manager*

Project manager memastikan proyek berjalan sesuai rencana, waktu, biaya, risiko, dan kualitas.

Tanggung jawab:

- menyusun rencana kerja;
- mengelola jadwal;
- mengelola risiko;
- mengkoordinasikan tim;
- mengatur rapat;
- memantau progres;
- mengelola isu;
- memastikan dokumentasi lengkap;
- menjaga komunikasi dengan sponsor.

Prinsip kritis: proyek DW tanpa manajemen proyek yang baik akan mudah berubah menjadi daftar permintaan tanpa akhir.

12.2 Struktur Tim Kecil

Organisasi kecil sering tidak mampu membentuk tim lengkap. Solusinya bukan menghapus fungsi, tetapi **merangkap peran secara sadar** dan tetap menjaga kontrol penting.

Contoh struktur tim kecil:

Peran	Bisa Dirangkap?	Fokus
Data Architect	Ya	desain model dan arsitektur
Data Engineer	Tidak disarankan	pipeline ETL/ELT
DBA / Platform Engineer	Ya	database, backup, tuning
BI Developer	Ya	dashboard
Data Steward	Ya	kualitas dan definisi data
Project Manager	Ya	koordinasi

A. Pola Tim Kecil 3 Orang

Contoh:

Orang	Peran Dirangkap
Orang 1	*Data Architect* + *Project Manager* + *Product Owner Data*
Orang 2	*Data Engineer* + *ETL Developer*
Orang 3	DBA / *Platform Engineer* + *BI Developer* + *Data Steward*

Risiko:

- beban terlalu besar;
- dokumentasi sering tertinggal;
- keamanan bisa kurang kuat;
- tidak ada reviewer independen;
- *data engineer* menjadi titik gagal tunggal.

Mitigasi:

- gunakan checklist;
- wajibkan dokumentasi;
- buat review mingguan;
- libatkan *data owner* dari bisnis;
- gunakan UAT formal;
- batasi *scope* MVP.

B. Pola Tim Kecil 5 Orang

Lebih sehat:

Peran	Jumlah
Sponsor / Product Owner	1
Data Architect	1
Data Engineer	1
DBA / Platform Engineer	1
BI Developer + Data Analyst	1

Tambahkan *data steward* dari unit bisnis sebagai peran paruh waktu.

Prinsip kritis: tim kecil harus memilih *scope* kecil. Jangan meminta tim kecil membangun DW enterprise penuh.

12.3 Struktur Tim Menengah

Organisasi menengah membutuhkan struktur lebih formal karena sumber data lebih banyak, pengguna lebih banyak, dan risiko lebih tinggi.

Struktur minimum:

- 1 *Data Architect*.
- 2–3 *Data Engineer*.
- 1 DBA.
- 1 *BI Developer*.
- 1 *Data Steward*.
- 1 PM.
- 1 *Security Reviewer*.

A. Contoh Struktur Tim Menengah

Peran	Jumlah	Fokus
Sponsor bisnis	1	mandat dan keputusan
Product Owner Data	1	prioritas kebutuhan
Data Architect	1	arsitektur dan model
Data Engineer	2–3	pipeline dan integrasi
DBA / *Platform Engineer*	1	platform, backup, performa
BI Developer	1	dashboard dan laporan
Data Steward	1	definisi, kualitas, metadata
Security Reviewer	1	akses, masking, audit
PM	1	koordinasi, risiko, jadwal

B. Kapan Struktur Menengah Dibutuhkan?

Gunakan struktur menengah jika:

- ada lebih dari 3 sumber data;
- ada lebih dari 2 domain bisnis;
- ada dashboard untuk banyak unit;
- ada kebutuhan audit;
- ada data sensitif;
- ada refresh harian;
- ada target produksi formal;
- ada kebutuhan integrasi historis.

C. Risiko Tim Menengah

Risiko:

- koordinasi mulai kompleks;
- definisi KPI antar unit bisa berbeda;
- backlog bisa membesar;
- tim teknis dan bisnis bisa tidak sinkron;
- kebutuhan keamanan lebih rumit.

Mitigasi:

- gunakan *project charter*;
- buat *data governance forum* kecil;
- tetapkan *data owner* per domain;
- gunakan *access matrix*;
- buat dokumentasi wajib;
- lakukan UAT per domain.

12.4 Struktur Tim *Enterprise*

Pada skala enterprise, DW bukan lagi proyek kecil. Ia menjadi **kapabilitas organisasi**. Tim harus dipisah berdasarkan fungsi, bukan hanya individu.

Struktur enterprise:

- *Data platform team*.
- *Data governance office*.
- *Data engineering squad*.
- *BI & analytics squad*.
- *Security & compliance*.
- *FinOps / cost controller*.
- *Data quality team*.

EDM Council menjelaskan DCAM sebagai kerangka praktik terbaik untuk manajemen data dan analitik lanjutan, yang mendefinisikan kapabilitas untuk membangun, menjalankan, dan mempertahankan disiplin manajemen data yang matang. (EDM Council)

A. *Data Platform Team*

Fokus:

- platform database;
- storage;
- compute;
- monitoring;
- backup;
- *disaster recovery*;
- performa;
- keamanan platform;
- otomatisasi deployment.

Peran:

- platform lead;
- DBA;
- infrastructure engineer;
- observability engineer;
- backup / DR engineer.

B. *Data Governance Office*

Fokus:

- kebijakan data;
- *data ownership*;
- data standard;
- *data dictionary*;
- metadata;
- *data lineage*;
- kualitas data;
- compliance;
- review akses.

Peran:

- head of data governance;
- data governance lead;
- data steward per domain;
- metadata manager;
- data quality lead.

DAMA-DMBOK menekankan bahwa pendekatan manajemen data yang terstruktur membantu organisasi mengelola aset data agar selaras dengan strategi bisnis dan kepatuhan. (DAMA International®)

C. Data Engineering Squad

Fokus:

- ingestion;
- *ETL/ELT*;
- CDC;
- *data pipeline*;
- staging;
- transformasi;
- data mart;
- testing pipeline;
- automation.

Peran:

- squad lead;
- senior data engineer;
- data engineer;
- ETL developer;
- data reliability engineer.

D. BI & Analytics Squad

Fokus:

- dashboard;
- laporan;
- semantic layer;
- analitik bisnis;
- self-service analytics;
- UAT dashboard;
- pelatihan pengguna.

Peran:

- BI lead;
- BI developer;
- data analyst;
- analytics engineer;
- dashboard designer.

E. *Security & Compliance*

Fokus:

- akses;
- klasifikasi data;
- masking;
- audit log;
- enkripsi;
- review risiko;
- kepatuhan regulasi;
- insiden keamanan.

Peran:

- security officer;
- compliance officer;
- auditor teknologi;
- privacy reviewer.

F. *FinOps / Cost Controller*

Pada platform modern, biaya compute, storage, backup, dan transfer data perlu dikontrol. Peran ini memastikan DW tidak hanya cepat, tetapi juga ekonomis.

Fokus:

- estimasi biaya;
- monitoring biaya berjalan;
- biaya per domain;
- biaya per workload;
- optimasi storage;
- optimasi compute;
- kebijakan retensi;
- laporan biaya ke sponsor.

Prinsip kritis: DW enterprise tanpa pengendalian biaya dapat berubah menjadi sistem strategis yang mahal tetapi tidak efisien.

G. *Data Quality Team*

Fokus:

- aturan kualitas data;
- *data profiling*;
- validasi;
- monitoring error;
- root cause analysis;
- perbaikan sumber data;
- laporan kualitas data;
- koordinasi dengan *data owner*.

Peran:

- data quality lead;
- data quality analyst;
- data steward;
- data tester.

12.5 Kompetensi Sertifikasi

Peserta sertifikasi *Data Warehouse* perlu menguasai kompetensi lintas bisnis, teknis, tata kelola, keamanan, biaya, dan operasi.

Peserta tidak harus menjadi ahli semua bidang, tetapi harus mampu memahami **cara kerja end-to-end** dan membuat keputusan desain yang benar.

Kompetensi utama:

- mendesain arsitektur;
- membaca kebutuhan bisnis;
- membuat model dimensional;
- merancang *ETL/ELT*;
- mengelola metadata;
- menjamin kualitas data;
- menghitung kapasitas;
- mengelola keamanan;
- menghitung biaya;
- menyusun rencana operasi.

A. Mendesain Arsitektur

Peserta harus mampu menggambar alur:

```
Source System → Staging → ETL/ELT → Data Warehouse Core →
Data Mart → Dashboard
```

Harus memahami:

- *source system*;
- *staging area*;
- DW core;
- *data mart*;
- metadata repository;
- OLAP layer;
- dashboard;
- governance layer;
- security layer.

B. Membaca Kebutuhan Bisnis

Peserta harus mampu menerjemahkan kebutuhan bisnis menjadi kebutuhan data.

Harus bisa menjawab:

- laporan apa yang dibutuhkan?
- keputusan apa yang didukung?
- siapa pengguna utama?
- KPI apa yang dipakai?
- data apa yang diperlukan?
- frekuensi refresh berapa?
- risiko apa yang muncul?

C. Membuat Model Dimensional

Peserta harus mampu:

- menentukan *grain*;
- membuat *fact table*;
- membuat *dimension table*;
- menentukan *measure*;
- membuat *star schema*;
- memahami *snowflake schema*;
- memilih SCD Type 0, 1, 2, atau 3;
- menggunakan *surrogate key*.

D. Merancang *ETL/ELT*

Peserta harus memahami:

- *full extract*;
- *incremental extract*;
- CDC;
- API ingestion;
- file ingestion;
- cleansing;
- standardisasi;
- deduplikasi;
- data masking;
- error handling;
- logging;
- rekonsiliasi.

E. Mengelola Metadata

Peserta harus mampu membuat:

- *data dictionary*;
- *ETL mapping*;
- *data lineage*;
- definisi KPI;
- metadata teknis;
- metadata operasional;
- ownership data.

Referensi metadata menyebut bahwa metadata membantu integrasi, performa query, lineage, governance, compliance, kualitas data, dokumentasi, dan skalabilitas DW.

F. Menjamin Kualitas Data

Peserta harus mampu memeriksa:

- *completeness*;
- *accuracy*;
- *consistency*;
- *timeliness*;
- *uniqueness*;
- *validity*.

Harus mampu membuat checklist:

- field penting kosong;
- duplikasi;
- format tanggal;
- kode referensi;
- nilai transaksi tidak wajar;
- data terlambat;
- selisih sumber vs target.

G. Menghitung Kapasitas

Peserta harus mampu menghitung:

> Ukuran Fact = jumlah baris × ukuran rata-rata baris
> Storage Total = Raw + Staging + Warehouse + Index + Backup + Retention

Harus memahami:

- growth rate;
- retensi;
- backup;
- indeks;
- staging;
- agregasi;
- storage cadangan;
- compute;
- concurrency.

H. Mengelola Keamanan

Peserta harus mampu membuat:

- klasifikasi data;
- *access matrix*;
- RBAC;
- masking;
- audit log;
- kontrol ekspor;
- review akses;
- perlindungan data sensitif.

I. Menghitung Biaya

Peserta harus mampu menghitung:

- biaya SDM;
- biaya platform;
- biaya storage;
- biaya backup;
- biaya operasional;
- biaya pelatihan;
- biaya maintenance;
- biaya risiko.

J. Menyusun Rencana Operasi

Peserta harus mampu membuat:

- checklist operasi harian;
- checklist monitoring;
- jadwal backup;
- restore test;
- rencana DR;
- maintenance plan;
- prosedur eskalasi;
- laporan insiden.

12.6 Estimasi Biaya SDM

Biaya SDM harus dihitung sejak awal. Banyak organisasi hanya menghitung biaya teknologi, tetapi lupa bahwa biaya terbesar sering muncul dari analisis, desain, ETL, testing, dokumentasi, UAT, operasi, dan perubahan kebutuhan.

Rumus dasar:

> Biaya SDM = jumlah orang × durasi bulan × biaya per orang per bulan

Untuk perhitungan profesional, tambahkan:

> Total Biaya SDM = biaya inti + biaya paruh waktu + biaya pelatihan + cadangan risiko

A. Contoh Estimasi Biaya SDM Proyek DW

Peran	Jumlah	Durasi	Biaya/Bulan	Total
Data Architect	1	3 bulan	Rp35 juta	Rp105 juta
Data Engineer	2	4 bulan	Rp25 juta	Rp200 juta
BI Developer	1	3 bulan	Rp20 juta	Rp60 juta
DBA / *Platform Engineer*	1	2 bulan	Rp25 juta	Rp50 juta
PM	1	4 bulan	Rp25 juta	Rp100 juta
Total				**Rp515 juta**

B. Tambahan Biaya yang Sering Terlupakan

Biaya yang sering tidak dihitung:

Komponen	Dampak
Data steward paruh waktu	kualitas data dan definisi KPI
Security reviewer	akses, masking, audit
UAT user bisnis	waktu pengguna untuk validasi
Pelatihan pengguna	adopsi dashboard
Dokumentasi	audit dan maintenance
Post-go-live support	stabilisasi produksi
Maintenance awal	perbaikan setelah produksi
Cadangan risiko	perubahan scope dan data buruk

Contoh tambahan:

Komponen	Estimasi
Data steward paruh waktu	Rp30 juta
Security review	Rp25 juta
UAT dan workshop bisnis	Rp20 juta
Pelatihan pengguna	Rp20 juta
Dokumentasi tambahan	Rp15 juta
Post-go-live support	Rp40 juta
Cadangan risiko 15% dari Rp515 juta	Rp77,25 juta

Estimasi total lebih realistis:

> Rp515 juta + Rp30 juta + Rp25 juta + Rp20 juta + Rp20 juta + Rp15 juta + Rp40 juta + Rp77,25 juta
> = Rp742,25 juta

Rekomendasi estimasi realistis: untuk proyek menengah, siapkan kisaran **Rp700–750 juta** jika memasukkan validasi bisnis, keamanan, dokumentasi, pelatihan, dan stabilisasi awal.

C. Estimasi Berdasarkan Fase

Fase	Aktivitas	SDM Dominan	Estimasi Porsi Biaya
Perencanaan	charter, requirement, scope	PM, architect, product owner	10–15%
Desain	arsitektur, model, security	architect, steward, security	15–20%
Implementasi	ETL, database, dashboard	engineer, DBA, BI developer	35–45%
Testing	data quality, UAT, performance	engineer, analyst, user bisnis	10–15%
Deployment	go-live, akses, backup	DBA, engineer, PM	5–10%
Stabilisasi	monitoring, perbaikan, training	semua peran kunci	10–15%

D. Estimasi Tim Kecil

Contoh proyek kecil 3 bulan:

Peran	Jumlah	Durasi	Biaya/Bulan	Total
Data Architect / PM	1	2 bulan	Rp30 juta	Rp60 juta
Data Engineer	1	3 bulan	Rp25 juta	Rp75 juta
DBA / BI Developer	1	2 bulan	Rp22 juta	Rp44 juta
Data Steward paruh waktu	1	2 bulan	Rp10 juta	Rp20 juta
Total				**Rp199 juta**

Cocok untuk:

- MVP;
- satu domain data;
- 1–2 sumber data;
- dashboard sederhana;
- data historis terbatas;
- risiko rendah.

E. Estimasi Tim Menengah

Contoh proyek 4 bulan:

Peran	Jumlah	Durasi	Biaya/Bulan	Total
Data Architect	1	4 bulan	Rp35 juta	Rp140 juta
Data Engineer	2	4 bulan	Rp25 juta	Rp200 juta
DBA / *Platform Engineer*	1	3 bulan	Rp25 juta	Rp75 juta
BI Developer	1	3 bulan	Rp20 juta	Rp60 juta
Data Steward	1	3 bulan	Rp18 juta	Rp54 juta
PM	1	4 bulan	Rp25 juta	Rp100 juta
Security Reviewer	1	1 bulan	Rp25 juta	Rp25 juta
Total				**Rp654 juta**

Tambahkan cadangan 15%:

Rp654 juta × 15% = Rp98,1 juta

Total:

Rp654 juta + Rp98,1 juta = Rp752,1 juta

F. Estimasi Tim Enterprise

Untuk enterprise, biaya SDM lebih baik dihitung per fungsi, bukan hanya per orang.

Fungsi	Estimasi Komponen
Platform	DBA, platform engineer, monitoring
Engineering	data engineer, ETL developer, automation
Governance	data governance, steward, metadata
BI & Analytics	BI developer, analyst, semantic layer
Security	security review, compliance, audit
PMO	PM, change management, training
FinOps	cost monitoring, capacity review

Pada skala enterprise, biaya SDM tidak berhenti saat implementasi selesai. Harus ada **biaya operasi tahunan**.

Contoh komponen operasi tahunan:

- monitoring;
- backup dan restore test;
- maintenance;
- review akses;
- penambahan sumber data;
- perbaikan kualitas data;
- pelatihan pengguna;
- cost optimization;
- audit support.

Hands-on. Template Perencanaan SDM dan Biaya *Data Warehouse*

Gunakan template ini untuk menyusun rencana SDM proyek DW.

A. Identitas Proyek

Elemen	Isi
Nama proyek	
Sponsor bisnis	
Domain data awal	
Target fase	MVP / Pilot / Produksi
Durasi proyek	
Jumlah sumber data	
Jumlah dashboard	
Risiko utama	

B. Matriks Peran dan Tanggung Jawab

Peran	Tanggung Jawab Utama	Nama / Unit	Full-time / Part-time
Sponsor bisnis	mandat dan anggaran		
Product owner data	prioritas kebutuhan		
Data architect	desain arsitektur dan model		
Data engineer	pipeline ETL/ELT		
DBA / *platform engineer*	database, backup, tuning		
BI developer	dashboard		
Data steward	definisi dan kualitas data		
Security reviewer	akses dan compliance		
PM	koordinasi proyek		

C. RACI Sederhana

Keterangan:

- R = *Responsible*
- A = *Accountable*
- C = *Consulted*
- I = *Informed*

Aktivitas	Sponsor	PO Data	Architect	Engineer	DBA	BI	Steward	Security	PM
Project charter	A	R	C	I	I	I	C	C	R
Requirement	C	A/R	C	C	I	C	C	I	R
Data modeling	I	C	A/R	C	C	I	C	I	I
ETL/ELT	I	C	C	A/R	C	I	C	I	I
Data quality	I	C	C	R	I	C	A/R	I	I
Security setup	I	C	C	C	R	I	C	A/R	I
Dashboard	I	A	C	C	I	R	C	I	I
UAT	A	R	C	C	I	C	C	C	R
Deployment	I	C	C	R	A/R	C	I	C	R
Operation	I	C	C	R	R	C	C	C	I

D. Matriks Kompetensi

Kompetensi	Architect	Engineer	DBA	BI	Steward	Analyst	Security	PM
Arsitektur DW	Tinggi	Sedang	Sedang	Rendah	Rendah	Rendah	Rendah	Rendah
Dimensional modeling	Tinggi	Sedang	Rendah	Sedang	Sedang	Sedang	Rendah	Rendah
ETL/ELT	Sedang	Tinggi	Sedang	Rendah	Rendah	Rendah	Rendah	Rendah
Database tuning	Sedang	Sedang	Tinggi	Rendah	Rendah	Rendah	Rendah	Rendah
Dashboard	Rendah	Rendah	Rendah	Tinggi	Rendah	Sedang	Rendah	Rendah
Data quality	Sedang	Sedang	Rendah	Rendah	Tinggi	Sedang	Rendah	Rendah
Metadata	Tinggi	Sedang	Rendah	Rendah	Tinggi	Sedang	Rendah	Rendah
Security	Sedang	Sedang	Sedang	Rendah	Rendah	Rendah	Tinggi	Rendah
Cost planning	Sedang	Rendah	Sedang	Rendah	Rendah	Rendah	Sedang	Tinggi
Project control	Rendah	Rendah	Rendah	Rendah	Rendah	Rendah	Rendah	Tinggi

E. Template Estimasi Biaya SDM

Peran	Jumlah	Durasi Bulan	Biaya / Bulan	Total
Sponsor bisnis				
Product owner data				
Data architect				
Data engineer				
ETL developer				
DBA / *platform engineer*				
BI developer				
Data analyst				
Data steward				
Security reviewer				
PM				
Total				

F. Template Biaya Tambahan

Komponen	Estimasi	Catatan
Pelatihan pengguna		
UAT workshop		
Dokumentasi		
Post-go-live support		
Security review		
Audit support		
Cadangan risiko		
Total Tambahan		

G. Checklist Kesiapan SDM

Pertanyaan	Ya/Tidak
Apakah sponsor bisnis sudah jelas?	
Apakah *data owner* tiap domain sudah ditetapkan?	
Apakah *product owner data* tersedia?	
Apakah ada *data architect*?	
Apakah ada *data engineer* yang fokus?	
Apakah ada DBA / *platform engineer*?	
Apakah ada *BI developer*?	
Apakah ada *data steward*?	
Apakah ada reviewer keamanan?	
Apakah ada PM?	
Apakah peran boleh rangkap sudah disepakati?	
Apakah ada RACI?	
Apakah biaya SDM sudah dihitung?	
Apakah biaya operasi setelah go-live sudah dihitung?	

Kesalahan Umum Perencanaan SDM

Kesalahan yang sering terjadi:

- menganggap DW cukup dikerjakan oleh satu programmer;
- tidak menunjuk sponsor bisnis;
- tidak ada *data owner*;
- tidak ada *data steward*;
- *data engineer* merangkap semua hal;
- tim teknis membuat KPI tanpa persetujuan bisnis;
- keamanan baru dipikirkan setelah dashboard selesai;
- biaya UAT dan pelatihan tidak dihitung;
- dokumentasi tidak dialokasikan waktunya;
- tidak ada tim operasi setelah go-live;
- tidak ada cadangan risiko;
- struktur tim terlalu besar untuk MVP;
- struktur tim terlalu kecil untuk enterprise.

Prinsip kritis: kekurangan SDM tidak selalu terlihat saat proyek dimulai. Ia terlihat saat data salah, dashboard lambat, akses bocor, atau sistem gagal dipulihkan.

Ringkasan Bab

Perencanaan SDM adalah bagian penting dari proyek *Data Warehouse*. Tim harus mencakup fungsi bisnis, arsitektur, engineering, platform, BI, governance, keamanan, dan manajemen proyek. Pada organisasi kecil, beberapa peran dapat dirangkap, tetapi fungsi kritis tidak boleh hilang. Pada organisasi menengah dan enterprise, struktur harus lebih formal agar kualitas, keamanan, biaya, dan operasi dapat dikendalikan.

Poin penting:

- **DW adalah proyek organisasi, bukan hanya proyek teknologi.**
- Sponsor bisnis menentukan mandat dan prioritas.
- *Data architect* memastikan desain DW dapat tumbuh.
- *Data engineer* menjaga pipeline data.
- DBA / *platform engineer* menjaga performa, backup, dan operasi.
- *BI developer* membuat data dapat digunakan pengguna.
- *Data steward* menjaga definisi dan kualitas data.
- *Security reviewer* memastikan akses dan compliance.
- PM menjaga proyek tetap terkendali.
- Biaya SDM harus mencakup implementasi, UAT, dokumentasi, pelatihan, keamanan, dan stabilisasi.
- Untuk sertifikasi, peserta harus mampu memahami peran, kompetensi, biaya, dan struktur tim sesuai skala organisasi.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Mengapa *Data Warehouse* membutuhkan sponsor bisnis?
2. Apa perbedaan peran *data architect* dan *data engineer*?
3. Mengapa *data steward* penting untuk kualitas data?
4. Peran apa saja yang boleh dirangkap pada organisasi kecil?
5. Mengapa *data engineer* tidak disarankan terlalu banyak merangkap?
6. Apa perbedaan struktur tim kecil, menengah, dan enterprise?
7. Mengapa *security reviewer* perlu dilibatkan sejak awal?
8. Apa saja kompetensi wajib peserta sertifikasi *Data Warehouse*?
9. Bagaimana rumus dasar menghitung biaya SDM?
10. Mengapa biaya UAT, dokumentasi, dan post-go-live support perlu dihitung?

B. Studi Kasus

Sebuah organisasi ingin membangun *Data Warehouse* penjualan dan proyek dalam 4 bulan. Sumber data berasal dari transaksi penjualan, pelanggan, produk, cabang, proyek, progres mingguan, dan biaya proyek. Dashboard akan digunakan direksi, manajer unit, analis, dan auditor.

Tugas:

- tentukan struktur tim yang sesuai;
- tentukan peran yang boleh dirangkap;
- buat RACI sederhana;
- tentukan kompetensi yang harus ada;
- hitung estimasi biaya SDM;
- tambahkan cadangan risiko 15%;
- jelaskan risiko jika tidak ada *data steward* dan *security reviewer*.

C. Praktik

Buat rencana SDM proyek DW dengan format:

Area	Isi
Skala proyek	kecil / menengah / enterprise
Domain data	
Durasi proyek	
Struktur tim	
Peran yang dirangkap	
Kompetensi kritis	
RACI	
Estimasi biaya SDM	
Cadangan risiko	
Risiko SDM utama	
Mitigasi	

BAB 13. Perhitungan Biaya *Data Warehouse*

Data Warehouse tidak boleh hanya dihitung dari biaya server atau biaya penyimpanan. Biaya sebenarnya mencakup **SDM, storage, compute, backup, jaringan, keamanan, monitoring, lisensi, dukungan open source, konsultan, pelatihan, disaster recovery, dan biaya operasional bulanan**.

Kesalahan umum dalam proyek *Data Warehouse* adalah menghitung biaya terlalu optimistis. Pada awal proyek, biaya terlihat kecil. Setelah produksi, biaya mulai naik karena data bertambah, pengguna bertambah, dashboard bertambah, backup membesar, query makin kompleks, dan kebutuhan keamanan meningkat.

Referensi implementasi *Data Warehouse* menekankan bahwa proyek DW membutuhkan perencanaan, tim, *requirement gathering*, desain, ETL, konstruksi fisik, kualitas data, tuning, UAT, deployment, dan pemeliharaan. Artinya, biaya DW tidak hanya biaya teknologi, tetapi biaya **siklus hidup penuh**. FinOps Foundation menjelaskan FinOps sebagai kerangka operasional dan praktik budaya untuk memaksimalkan nilai bisnis teknologi, mempercepat keputusan berbasis data, dan membangun akuntabilitas finansial melalui kolaborasi tim teknik, keuangan, dan bisnis. (FinOps)

Golden Rule: biaya *Data Warehouse* yang benar bukan yang paling murah di awal, tetapi yang **paling terkendali sepanjang umur sistem dan paling selaras dengan nilai bisnis**.

13.1 Komponen Biaya

Komponen biaya *Data Warehouse* harus dihitung dari awal implementasi sampai operasi. Komponen ini dibagi menjadi biaya awal, biaya berjalan, biaya risiko, dan biaya pertumbuhan.

Komponen utama:

- perangkat keras / *cloud*;
- storage;
- compute;
- network;
- backup;
- lisensi software;
- *open source support*;
- SDM internal;
- konsultan;
- training;
- security;
- monitoring;
- *disaster recovery*;
- operasional bulanan.

A. Perangkat Keras / *Cloud*

Untuk pendekatan *on-premise*, biaya awal biasanya meliputi:

- server;
- storage;
- perangkat jaringan;
- perangkat backup;
- rack;
- UPS;
- pendingin;
- ruang fisik;
- instalasi;
- garansi perangkat.

Untuk pendekatan *cloud*, biaya biasanya berbasis pemakaian:

- storage bulanan;
- compute per jam / per kapasitas;
- transfer data;
- backup;
- monitoring;
- security service;
- support;
- biaya jaringan;
- biaya data keluar jika ada.

NIST menjelaskan *cloud computing* sebagai model akses jaringan yang nyaman dan sesuai permintaan ke kumpulan sumber daya komputasi seperti jaringan, server, storage, aplikasi, dan layanan yang dapat disediakan serta dilepas dengan cepat. NIST juga menempatkan *measured service* sebagai salah satu karakteristik penting, sehingga biaya *cloud* harus dipantau berdasarkan pemakaian. (NIST Computer Security Resource Center)

B. Storage

Storage tidak hanya untuk tabel utama. Dalam DW, storage mencakup:

- raw data;
- staging;
- data warehouse core;
- data mart;
- indeks;
- *materialized view*;
- backup;
- archive;
- log;
- temporary file;
- metadata.

Referensi storage DW menekankan bahwa strategi penyimpanan harus mempertimbangkan efisiensi, keandalan, biaya, scalability, fleksibilitas, keamanan, compliance, serta strategi hybrid dan archiving.

Contoh komponen storage:

Komponen	Fungsi
Raw data	arsip data mentah
Staging	area sementara validasi
Warehouse	data bersih dan terintegrasi
Data mart	data siap dashboard
Index	mempercepat query
Backup	pemulihan
Archive	retensi jangka panjang
Log	audit dan troubleshooting

Prinsip kritis: jangan menghitung storage hanya dari ukuran *fact table*. Itu hampir pasti terlalu rendah.

C. Compute

Compute adalah sumber daya untuk memproses *query*, ETL, transformasi, agregasi, refresh dashboard, dan analitik.

Biaya compute dipengaruhi oleh:

- jumlah user aktif;
- kompleksitas query;
- frekuensi refresh;
- volume data;
- concurrency;
- ETL harian;
- workload analitik;
- dashboard eksekutif;
- pemisahan workload.

Referensi performa DW menekankan bahwa monitoring perlu mencakup *query response time*, CPU, memory, storage, bandwidth, concurrency, queue length, wait time, dan error rate.

Contoh pembagian compute:

Workload	Karakter
ETL harian	berat, terjadwal
Dashboard	ringan-sedang, banyak pengguna
Analisis ad hoc	tidak terprediksi
Audit	historis, detail
Data quality check	periodik
Materialized view refresh	berat, terjadwal

D. Network

Biaya jaringan sering terlupakan.

Komponen jaringan:

- bandwidth internal;
- koneksi antar sistem sumber;
- transfer data dari cabang;
- transfer data keluar;
- koneksi ke dashboard;
- replikasi ke *secondary site*;
- sinkronisasi backup;
- akses VPN / private network jika ada.

Risiko biaya jaringan:

- data historis besar dipindahkan berulang;
- backup dikirim ke lokasi lain;
- dashboard membaca data terlalu besar;
- pipeline mengambil data penuh setiap hari;
- tidak ada kompresi;
- tidak ada incremental load.

E. Backup

Backup adalah biaya wajib, bukan opsional.

Komponen backup:

- storage backup;
- salinan backup;
- backup metadata;
- backup script ETL;
- backup konfigurasi dashboard;
- backup log audit;
- restore test;
- lokasi cadangan;
- retensi backup.

Referensi DW menjelaskan bahwa database backup adalah mekanisme membuat salinan data pada perangkat sekunder atau tersier secara berkala untuk menghindari dampak kerusakan atau korupsi data.

Prinsip kritis: biaya backup harus dihitung sejak awal. Backup dua salinan berarti biaya storage bisa naik signifikan.

F. Lisensi Software

Komponen lisensi dapat mencakup:

- database engine;
- alat ETL;
- alat BI;
- monitoring;
- backup;
- security;
- metadata catalog;
- data quality tool;
- scheduler;
- support enterprise.

Untuk buku ini, pendekatan yang direkomendasikan adalah **mengutamakan open source** jika sesuai kebutuhan, kompetensi SDM, keamanan, dan SLA. Namun, open source bukan berarti nol biaya. Tetap ada biaya instalasi, konfigurasi, dukungan, pelatihan, dokumentasi, dan maintenance.

G. *Open Source Support*

Biaya *open source support* dapat mencakup:

- jasa instalasi;
- hardening keamanan;
- tuning performa;
- support komunitas berbayar / vendor independen;
- training;
- dokumentasi;
- upgrade;
- bug fixing;
- integrasi;
- support operasional.

Prinsip kritis: *open source* mengurangi ketergantungan lisensi, tetapi tetap membutuhkan **kompetensi dan disiplin operasi**.

H. SDM Internal

Biaya SDM internal mencakup waktu orang yang terlibat dalam proyek.

Peran yang perlu dihitung:

- sponsor bisnis;
- *product owner data*;
- *data architect*;
- *data engineer*;
- DBA / *platform engineer*;
- *BI developer*;
- *data analyst*;
- *data steward*;
- security reviewer;
- auditor;
- PM;
- user UAT.

Referensi implementasi DW menegaskan bahwa proyek membutuhkan kolaborasi antara stakeholder bisnis, tim TI, data architect, administrator database, dan analis.

I. Konsultan

Konsultan dapat dibutuhkan untuk:

- arsitektur;
- audit desain;
- implementasi awal;
- ETL kompleks;
- tuning performa;
- security review;
- governance;
- dokumentasi sertifikasi;
- training;
- post-go-live support.

Gunakan konsultan untuk mempercepat dan mengurangi risiko, tetapi jangan membuat organisasi bergantung total. Harus ada transfer knowledge.

J. Training

Training diperlukan untuk:

- tim teknis;
- data steward;
- pengguna dashboard;
- auditor;
- manajer unit;
- tim keamanan;
- tim operasi.

Biaya training mencakup:

- materi;
- fasilitator;
- waktu peserta;
- lab;
- dokumentasi;
- evaluasi;
- sertifikasi jika ada.

K. Security

Biaya keamanan mencakup:

- klasifikasi data;
- RBAC;
- audit log;
- masking;
- enkripsi;
- review akses;
- penetration test jika diperlukan;
- hardening;
- compliance review;
- incident response plan.

Jangan menunda biaya keamanan. Biaya insiden biasanya lebih besar daripada biaya pencegahan.

L. Monitoring

Monitoring mencakup:

- monitoring ETL;
- monitoring database;
- monitoring dashboard;
- monitoring query lambat;
- monitoring storage;
- monitoring backup;
- alerting;
- log retention;
- observability dashboard.

Monitoring adalah biaya operasi yang menjaga sistem tetap sehat. Tanpa monitoring, organisasi hanya tahu masalah setelah pengguna mengeluh.

M. *Disaster Recovery*

Biaya DR mencakup:

- *secondary site*;
- replikasi;
- backup offsite;
- restore test;
- failover drill;
- dokumentasi DR;
- latihan insiden;
- kapasitas cadangan;
- SDM yang siap merespons.

RPO dan RTO menentukan biaya. Semakin kecil toleransi kehilangan data dan semakin cepat target pemulihan, semakin mahal desainnya.

N. Operasional Bulanan

Biaya operasional bulanan mencakup:

- compute;
- storage;
- backup;
- bandwidth;
- support;
- SDM operasi;
- monitoring;
- maintenance;
- security review;
- dashboard support;
- data quality correction.

Prinsip kritis: biaya terbesar DW sering bukan saat dibangun, tetapi saat **dipakai, diperluas, dan dipertahankan**.

13.2 CAPEX vs OPEX

Dalam perencanaan biaya, organisasi perlu memahami perbedaan CAPEX dan OPEX.

A. CAPEX

CAPEX atau *capital expenditure* adalah belanja modal. Biasanya dibayar di awal dan digunakan beberapa tahun.

Contoh CAPEX:

- server;
- storage;
- perangkat jaringan;
- perangkat backup;
- rack;
- UPS;
- fasilitas ruang server;
- instalasi awal;
- lisensi perpetual jika ada.

Kelebihan CAPEX:

- aset dimiliki organisasi;
- biaya awal jelas;
- kontrol fisik lebih tinggi;
- cocok untuk beban stabil;
- cocok jika regulasi menuntut kontrol lokal.

Kekurangan CAPEX:

- biaya awal besar;
- kapasitas sering harus dibeli di depan;
- scaling lambat;
- risiko over-provisioning;
- perlu tim operasi kuat;
- biaya listrik dan pendingin harus dihitung.

B. OPEX

OPEX atau *operational expenditure* adalah biaya operasional berjalan.

Contoh OPEX:

- layanan *cloud*;
- listrik;
- pendingin;
- bandwidth;
- support;
- SDM operasi;
- monitoring;
- backup bulanan;
- maintenance;
- training berkala;
- biaya keamanan;
- biaya konsultan operasional.

Kelebihan OPEX:

- lebih fleksibel;
- kapasitas bisa mengikuti kebutuhan;
- biaya awal lebih rendah;
- cocok untuk pertumbuhan cepat;
- cocok untuk eksperimen dan MVP.

Kekurangan OPEX:

- biaya bisa membengkak jika tidak dikontrol;
- perlu monitoring biaya;
- sulit diprediksi jika workload tidak disiplin;
- risiko ketergantungan platform;
- biaya data transfer dan backup sering terlupakan.

FinOps Foundation menekankan optimasi penggunaan dan biaya sebagai upaya menciptakan nilai dengan merancang sistem secara efektif, mengurangi pemborosan, dan mengoptimalkan tarif sumber daya. (FinOps)

C. Perbandingan CAPEX dan OPEX

Aspek	CAPEX	OPEX
Pola biaya	besar di awal	bulanan / pemakaian
Kepemilikan	aset sendiri	layanan / konsumsi
Scaling	lebih lambat	lebih cepat
Risiko	salah beli kapasitas	biaya berjalan membengkak
Cocok untuk	beban stabil	beban dinamis
Kontrol	tinggi	tergantung desain
Tantangan	operasi fisik	*cost governance*

Prinsip kritis: CAPEX tidak selalu lebih murah. OPEX tidak selalu lebih mahal. Yang menentukan adalah pola beban, pertumbuhan, SDM, risiko, SLA, dan disiplin biaya.

13.3 Rumus TCO

TCO atau *Total Cost of Ownership* adalah total biaya kepemilikan selama periode tertentu. Untuk DW, TCO harus mencakup biaya implementasi dan biaya operasi.

Rumus dasar:

> TCO 3 Tahun = biaya implementasi awal + total biaya operasional 36 bulan

Rumus lebih lengkap:

> TCO 3 Tahun =
> Biaya Implementasi Awal
> + (Biaya Operasional Bulanan × 36)
> + Biaya Training
> + Biaya Security
> + Biaya DR
> + Biaya Support
> + Cadangan Risiko

Komponen Implementasi Awal

Komponen	Contoh
Analisis kebutuhan	workshop, requirement brief
Arsitektur	desain DW, data mart, security
SDM proyek	architect, engineer, BI, DBA, PM
ETL	extract, transform, load
Modeling	fact, dimension, SCD
Dashboard	desain dan implementasi
Testing	data quality, UAT, performance
Deployment	migrasi, initial load, sign-off
Dokumentasi	data dictionary, lineage, manual

Komponen Operasional Bulanan

Komponen	Contoh
Storage	warehouse, raw, staging
Compute	ETL, dashboard, analyst query
Backup	salinan, archive, restore test
Network	transfer data, replikasi
Monitoring	alert, log, observability
Support	incident, bug, tuning
Security	review akses, audit log
SDM operasi	engineer, DBA, steward

Cadangan Risiko

Tambahkan cadangan risiko untuk:

- data sumber buruk;
- scope bertambah;
- perubahan KPI;
- integrasi tambahan;
- dashboard baru;
- peningkatan storage;
- tuning performa;
- kebutuhan audit;
- training tambahan.

Rekomendasi awal:

> Cadangan Risiko = 10%–20% dari total estimasi

Untuk proyek dengan data banyak, sumber tidak stabil, atau governance lemah, gunakan cadangan lebih tinggi.

13.4 Rumus Biaya *Cloud DW*

Rumus:

> Biaya Cloud =
> biaya storage
> + biaya compute
> + biaya transfer data
> + biaya backup
> + biaya monitoring

Rumus lebih lengkap:

> Biaya Cloud Bulanan =
> Storage Aktif
> + Raw / Archive Storage
> + Compute ETL
> + Compute BI
> + Compute Analyst
> + Data Transfer
> + Backup
> + Monitoring
> + Security
> + Support

A. Biaya Storage

> Biaya Storage = total TB tersimpan × biaya per TB per bulan

Komponen:

- warehouse;
- raw archive;
- staging;
- data mart;
- backup;
- log;
- metadata;
- temporary storage.

B. Biaya Compute

Biaya Compute = jam compute per bulan × biaya compute per jam

Pisahkan:

Workload	Contoh Perhitungan
ETL	2 jam/hari × 30 hari
Dashboard	10 jam/hari × 22 hari
Analyst	4 jam/hari × 22 hari
Data quality	1 jam/hari × 30 hari
Maintenance	4 jam/bulan

Referensi Snowflake yang digunakan menekankan pengelolaan compute seperti auto suspend, auto resume, workload pattern, query timeout, dan monitoring penggunaan sumber daya sebagai cara mengendalikan biaya dan performa.

C. Biaya Transfer Data

Biaya Transfer = volume data keluar × biaya per GB

Yang harus diperhatikan:

- transfer dari sumber ke DW;
- transfer antar region;
- transfer ke dashboard;
- transfer backup;
- ekspor data;
- replikasi ke DR.

Biaya transfer sering kecil di awal, tetapi membesar saat data historis dan integrasi bertambah.

D. Biaya Backup

Biaya Backup = ukuran backup × jumlah salinan × biaya storage backup

Jika backup dua salinan:

Backup = data yang dibackup × 2

Jangan lupa:

- metadata;
- script ETL;
- konfigurasi dashboard;
- log audit;
- file raw penting.

E. Biaya Monitoring

Monitoring dapat dihitung sebagai:

Biaya Monitoring = biaya tool + storage log + alerting + SDM review

Jika monitoring tidak dihitung, sistem mungkin murah tetapi buta. Sistem buta berisiko tinggi.

13.5 Rumus Biaya *On-Premise DW*

Rumus:

> Biaya On-Premise =
> server
> + storage
> + jaringan
> + listrik
> + pendingin
> + lisensi
> + SDM
> + maintenance

Rumus lebih lengkap:

> TCO On-Premise =
> CAPEX Infrastruktur
> + Instalasi
> + Lisensi / Support
> + Listrik
> + Pendingin
> + Ruang
> + Bandwidth
> + Backup
> + DR
> + SDM Operasi
> + Maintenance
> + Cadangan Risiko

A. Server

Komponen:

- CPU;
- RAM;
- storage internal;
- network interface;
- redundancy;
- garansi;
- spare part.

Biaya server harus mengikuti beban:

- ETL;
- query dashboard;
- analisis;
- indexing;

- backup;
- restore;
- concurrency.

B. Storage

Komponen:

- storage utama;
- storage backup;
- storage archive;
- storage log;
- storage temporary;
- storage cadangan.

Gunakan perhitungan dari Bab 10:

> Storage Total = Raw + Staging + Warehouse + Index + Backup + Retention

C. Jaringan

Komponen:

- switch;
- router;
- kabel;
- bandwidth;
- replikasi;
- firewall;
- koneksi cabang;
- koneksi DR.

D. Listrik dan Pendingin

Biaya listrik:

> Biaya Listrik = daya kW × jam operasi × tarif per kWh

Contoh:

> Daya = 2 kW
> Jam operasi = 24 × 30 = 720 jam
> Tarif = Rp1.500/kWh
>
> Biaya = 2 × 720 × 1.500
> = Rp2.160.000/bulan

Pendingin biasanya dihitung sebagai overhead fasilitas. Bisa memakai estimasi sederhana:

> Biaya Pendingin = 30%–100% dari biaya listrik IT

Angka aktual tergantung efisiensi ruang server.

E. Maintenance

> Maintenance mencakup:

- kontrak support perangkat;
- penggantian disk;
- update sistem;
- patch keamanan;
- backup media;
- monitoring;
- tuning;
- dokumentasi;
- DR drill.

13.6 Rumus Biaya per *Query*

Biaya per *query* membantu mengukur efisiensi pemakaian sistem.

Rumus:

> Biaya per Query = total biaya bulanan / jumlah query bulanan

Contoh:

> Total biaya bulanan = Rp100.000.000
> Jumlah query bulanan = 2.000.000 query
>
> Biaya per query = Rp100.000.000 / 2.000.000
> = Rp50/query

Kapan Metrik Ini Berguna?

Berguna untuk:

- membandingkan efisiensi bulan ke bulan;
- melihat dampak query berat;
- mengukur biaya analitik;
- mendeteksi pemborosan;
- menilai optimasi materialized view;
- mengatur workload.

Namun, metrik ini tidak boleh dipakai sendirian. Query kecil dan query besar tidak sama. Lebih baik kelompokkan:

Jenis Query	Metrik
dashboard ringan	biaya per view
laporan standar	biaya per laporan
query analis	biaya per query berat
ETL	biaya per batch
audit	biaya per pemeriksaan

13.7 Rumus Biaya per Laporan

Biaya per laporan membantu melihat apakah laporan aktif benar-benar bernilai.

Rumus:

> Biaya per Laporan = total biaya bulanan / jumlah laporan aktif

Contoh:

> Total biaya bulanan = Rp100.000.000
> Jumlah laporan aktif = 50
>
> Biaya per laporan = Rp100.000.000 / 50
> = Rp2.000.000/laporan/bulan

Interpretasi

Jika ada 50 laporan tetapi hanya 10 yang dipakai, biaya efektif per laporan aktif menjadi lebih tinggi.

Rumus lebih tajam:

> Biaya per Laporan Aktif =
> total biaya bulanan / jumlah laporan yang dipakai minimal 1x per bulan

Contoh:

> Rp100.000.000 / 10 laporan aktif
> = Rp10.000.000/laporan aktif/bulan

Prinsip kritis: laporan yang tidak dipakai tetap memakan biaya: storage, refresh, maintenance, testing, dan support.

13.8 Contoh Simulasi Biaya

Skenario:

- 5 TB data awal;
- growth 30% per tahun;
- 50 user BI;
- 10 user aktif bersamaan;
- retensi 5 tahun;
- backup 2 salinan;
- refresh harian;
- estimasi TCO 3 tahun.

A. Hitung Pertumbuhan Data 3 Tahun

Data awal = 5 TB.
Growth = 30% per tahun.

Tahun	Data Aktif	Rumus
Tahun 1	5,00 TB	5
Tahun 2	6,50 TB	5 × 1,3
Tahun 3	8,45 TB	6,5 × 1,3

Total data aktif tahun ke-3:

8,45 TB

Untuk TCO 3 tahun, rata-rata data aktif dapat dihitung sederhana:

Rata-rata Data Aktif = (5 + 6,5 + 8,45) / 3
= 6,65 TB

B. Tambahkan Komponen Storage

Asumsi:

- raw archive = 100% data aktif;
- staging = 20% data aktif;
- index + materialized view = 30% data aktif;
- backup = 2 salinan dari data aktif + index;
- log dan metadata = 5%;
- cadangan = 25%.

Menggunakan data tahun ke-3: 8,45 TB.

Komponen	Rumus	Hasil
Data aktif DW	data tahun ke-3	8,45 TB
Raw archive	100% × data aktif	8,45 TB
Staging	20% × data aktif	1,69 TB
Index + materialized view	30% × data aktif	2,54 TB
Backup 2 salinan	2 × (data aktif + index)	21,98 TB
Log + metadata	5% × data aktif	0,42 TB

Total sebelum cadangan:

8,45 + 8,45 + 1,69 + 2,54 + 21,98 + 0,42
= 43,53 TB

Cadangan 25%:

43,53 × 25% = 10,88 TB

Total storage tahun ke-3:

43,53 + 10,88 = 54,41 TB

Rekomendasi kapasitas storage: minimal **55 TB**, lebih aman dibulatkan menjadi **60 TB**.

C. Estimasi Compute

Skenario:

- 50 user BI;
- 10 user aktif bersamaan;
- rata-rata 3 query per user saat jam sibuk;
- kompleksitas rata-rata = 2;
- ETL harian = sedang;
- refresh harian.

Rumus:

> Beban Query = user aktif bersamaan × query per user × kompleksitas
> Beban Query = 10 × 3 × 2 = 60

Tambahkan skor ETL sedang:

> Skor ETL = 50

Total beban compute:

> Total Beban = 60 + 50 = 110

Interpretasi:

- beban sedang;
- perlu indeks dan agregasi;
- ETL sebaiknya di luar jam sibuk;
- dashboard membaca data mart, bukan tabel detail;
- perlu monitoring durasi query dan ETL.

D. Simulasi Biaya Bulanan

Gunakan angka asumsi untuk latihan. Angka nyata harus disesuaikan dengan harga lokal, desain, SLA, dan pilihan platform.

Komponen	Asumsi	Biaya/Bulan
Storage 60 TB	storage aktif + backup + archive	Rp30 juta
Compute ETL	refresh harian	Rp20 juta
Compute BI	dashboard dan query user	Rp25 juta
Monitoring	log, alert, dashboard teknis	Rp8 juta
Backup operation	pengelolaan dan restore test	Rp7 juta
Security operation	review akses, audit log	Rp10 juta
Support teknis	incident dan maintenance	Rp20 juta
Total OPEX/Bulan		**Rp120 juta**

E. Biaya Implementasi Awal

Contoh:

Komponen	Estimasi
Analisis kebutuhan	Rp75 juta
Desain arsitektur dan model	Rp125 juta
ETL dan integrasi	Rp250 juta
Dashboard	Rp100 juta
Security setup	Rp75 juta
Testing dan UAT	Rp75 juta
Dokumentasi	Rp50 juta
Deployment	Rp50 juta
Training	Rp50 juta
Total Implementasi Awal	**Rp850 juta**

F. Hitung TCO 3 Tahun

Rumus:

TCO 3 Tahun = biaya implementasi awal + total biaya operasional 36 bulan

TCO 3 Tahun = Rp850 juta + (Rp120 juta × 36)
= Rp850 juta + Rp4.320 juta
= Rp5.170 juta

Tambahkan cadangan risiko 15%:

Cadangan Risiko = Rp5.170 juta × 15%
= Rp775,5 juta

Total TCO realistis:

TCO 3 Tahun = Rp5.170 juta + Rp775,5 juta
= Rp5.945,5 juta

Estimasi TCO 3 tahun: sekitar **Rp5,95 miliar**.

G. Biaya per Query

Asumsi:

- total biaya bulanan = Rp120 juta;
- query bulanan = 1.200.000 query.

Biaya per Query = Rp120.000.000 / 1.200.000
= Rp100/query

Jika optimasi membuat compute turun 20%, biaya bulanan menjadi:

Rp120 juta × 80% = Rp96 juta

Biaya per query baru:

Rp96.000.000 / 1.200.000
= Rp80/query

Penghematan:

Rp20/query × 1.200.000 = Rp24 juta/bulan

H. Biaya per Laporan

Asumsi:

- total biaya bulanan = Rp120 juta;
- laporan aktif = 40.

Biaya per Laporan = Rp120.000.000 / 40
= Rp3.000.000/laporan/bulan

Jika hanya 20 laporan yang benar-benar dipakai:

Biaya per Laporan Aktif = Rp120.000.000 / 20
= Rp6.000.000/laporan aktif/bulan

Pelajaran: hapus, gabungkan, atau arsipkan laporan yang tidak dipakai.

Hands-on. Membuat Tabel Estimasi Biaya Data Warehouse 3 Tahun

Gunakan template berikut untuk menghitung estimasi biaya proyek DW.

A. Input Dasar

Parameter	Nilai
Data awal	
Growth per tahun	
Retensi	
Jumlah user BI	
User aktif bersamaan	
Refresh	harian / mingguan / bulanan
Backup salinan	
Index overhead	
Staging overhead	
Cadangan kapasitas	

B. Estimasi Pertumbuhan Data

Tahun	Data Aktif	Growth	Catatan
Tahun 1			
Tahun 2			
Tahun 3			

Rumus:

$$\text{Data Tahun ke-n} = \text{Data Tahun Sebelumnya} \times (1 + \text{growth rate})$$

C. Estimasi Storage

Komponen	Rumus	Hasil
Data aktif	data tahun ke-3	
Raw archive	% × data aktif	
Staging	% × data aktif	
Index + materialized view	% × data aktif	
Backup	salinan × data di backup	
Log + metadata	% × data aktif	
Cadangan	% × subtotal	
Total Storage	semua komponen	

D. Estimasi Compute

Komponen	Asumsi	Biaya/Bulan
ETL harian		
Dashboard BI		
Query analis		
Data quality check		
Materialized view refresh		
Maintenance		
Total Compute		

E. Estimasi OPEX Bulanan

Komponen	Biaya/Bulan
Storage	
Compute	
Network	
Backup	
Monitoring	
Security operation	
Support	
SDM operasi	
DR	
Total OPEX/Bulan	

F. Estimasi Implementasi Awal

Komponen	Biaya
Analisis kebutuhan	
Arsitektur	
Data modeling	
ETL/ELT	
Database setup	
Dashboard	
Security setup	
Testing	
UAT	
Deployment	
Training	
Dokumentasi	
Total Implementasi Awal	

G. Hitung TCO 3 Tahun

TCO 3 Tahun = Total Implementasi Awal + (Total OPEX/Bulan × 36)

Tambahkan cadangan risiko:

Cadangan Risiko = TCO 3 Tahun × 10% sampai 20%

Total akhir:

TCO Final = TCO 3 Tahun + Cadangan Risiko

H. Biaya per Query dan Laporan

Metrik	Rumus	Hasil
Biaya per Query	total biaya bulanan / query bulanan	
Biaya per Laporan	total biaya bulanan / laporan aktif	
Biaya per User	total biaya bulanan / user aktif	
Biaya per TB	total biaya bulanan / TB aktif	

I. Checklist *FinOps* DW

FinOps Foundation menekankan bahwa optimasi biaya harus dilakukan lintas peran, bukan hanya oleh tim keuangan. Aktivitas optimasi penggunaan dan biaya mencakup analisis resource agar sesuai pola pemakaian, efisiensi workload, dan nilai bisnis yang dihasilkan. (FinOps)

Pertanyaan	Ya/Tidak
Apakah biaya storage dipantau per domain data?	
Apakah biaya compute dipisahkan antara ETL, BI, dan analyst?	
Apakah dashboard yang tidak dipakai sudah dihapus / diarsipkan?	
Apakah query berat di monitor?	
Apakah data lama diarsipkan sesuai retensi?	
Apakah backup dihitung dalam TCO?	
Apakah biaya DR dihitung?	
Apakah biaya training dihitung?	
Apakah biaya security dihitung?	
Apakah ada pemilik biaya per domain?	
Apakah ada review biaya bulanan?	
Apakah optimasi biaya tidak mengorbankan kualitas dan audit?	

Kesalahan Umum Perhitungan Biaya

Kesalahan yang sering terjadi:

- hanya menghitung server atau storage utama;
- lupa menghitung backup;
- lupa menghitung raw dan staging;
- lupa menghitung indeks dan *materialized view*;
- tidak menghitung SDM internal;
- tidak menghitung UAT dan training;
- tidak menghitung security review;
- tidak menghitung monitoring;
- tidak menghitung DR;
- tidak menghitung biaya operasional bulanan;
- tidak menghitung pertumbuhan data;
- tidak menghitung laporan yang tidak dipakai;
- tidak menghitung biaya query berat;
- tidak membuat cadangan risiko.

Prinsip kritis: estimasi biaya yang terlalu murah di proposal sering menjadi biaya paling mahal saat produksi.

Ringkasan Bab

Perhitungan biaya *Data Warehouse* harus mencakup implementasi awal dan operasi jangka panjang. Komponen biaya meliputi perangkat keras atau *cloud*, storage, compute, network, backup, lisensi, *open source support*, SDM, konsultan, training, security, monitoring, DR, dan operasional bulanan. CAPEX cocok untuk aset dan kontrol fisik, sedangkan OPEX cocok untuk fleksibilitas dan pemakaian berjalan. TCO 3 tahun membantu organisasi melihat biaya penuh, bukan hanya biaya awal.

Poin penting:

- **Hitung biaya berdasarkan siklus hidup, bukan hanya pengadaan awal.**
- Storage harus mencakup raw, staging, warehouse, index, backup, log, archive, dan cadangan.
- Compute harus dipisahkan antara ETL, dashboard, analis, dan maintenance.
- Backup dan DR adalah biaya wajib.
- Open source tetap membutuhkan biaya support dan SDM.
- TCO 3 tahun memberi gambaran biaya realistis.
- Biaya per query dan biaya per laporan membantu mengukur efisiensi.
- FinOps membantu menghubungkan biaya teknologi dengan nilai bisnis.
- Laporan yang tidak dipakai tetap menimbulkan biaya.
- Cadangan risiko penting untuk perubahan scope, data buruk, dan kebutuhan audit.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Mengapa biaya DW tidak boleh dihitung hanya dari storage utama?
2. Jelaskan perbedaan CAPEX dan OPEX dalam konteks DW.
3. Apa saja komponen utama TCO 3 tahun?
4. Mengapa backup harus dihitung sebagai biaya terpisah?
5. Apa risiko biaya terbesar pada pendekatan *cloud DW*?
6. Apa risiko biaya terbesar pada pendekatan *on-premise DW*?
7. Bagaimana menghitung biaya per query?
8. Mengapa laporan yang tidak dipakai tetap harus masuk evaluasi biaya?
9. Mengapa *open source* tetap membutuhkan anggaran support?
10. Bagaimana FinOps membantu pengendalian biaya DW?

B. Studi Kasus

Sebuah organisasi memiliki skenario:

- data awal 5 TB;
- growth 30% per tahun;
- 50 user BI;
- 10 user aktif bersamaan;
- retensi 5 tahun;
- backup 2 salinan;
- refresh harian;
- biaya operasional bulanan diperkirakan Rp120 juta;
- biaya implementasi awal Rp850 juta.

Tugas:

- hitung data aktif tahun ke-3;
- hitung estimasi storage dengan raw, staging, index, backup, log, dan cadangan;
- hitung TCO 3 tahun;
- tambahkan cadangan risiko 15%;
- hitung biaya per query jika ada 1,2 juta query per bulan;
- hitung biaya per laporan jika ada 40 laporan aktif;
- berikan 5 rekomendasi optimasi biaya.

C. Praktik

Buat tabel estimasi biaya DW 3 tahun untuk organisasi Anda:

Area	Nilai
Data awal	
Growth	
Retensi	
Storage total	
Compute bulanan	
Backup bulanan	
Monitoring bulanan	
Security bulanan	
SDM operasi	
OPEX bulanan	
Implementasi awal	
TCO 3 tahun	
Cadangan risiko	
TCO final	
Biaya per query	
Biaya per laporan	
Rekomendasi optimasi	

Bab 14. *Environmentally Friendly Data Warehouse*

Data Warehouse modern tidak hanya dinilai dari kecepatan, kapasitas, dan keamanan. Semakin besar volume data, semakin sering *refresh*, semakin banyak *query*, dan semakin panjang retensi data, semakin besar pula konsumsi energi, storage, compute, backup, dan biaya operasional.

Karena itu, konsep *environmentally friendly Data Warehouse* bukan sekadar isu lingkungan. Ini adalah bagian dari **efisiensi arsitektur, disiplin data, pengendalian biaya, dan tata kelola teknologi**. Sistem yang boros compute biasanya juga boros biaya. Sistem yang menyimpan semua data tanpa retensi biasanya juga boros storage. Sistem yang menjalankan *query* buruk berulang kali biasanya juga boros energi.

Referensi yang digunakan menekankan bahwa performa *Data Warehouse* dapat ditingkatkan melalui *partitioning*, *indexing*, *materialized view*, kompresi, arsip data, optimasi *query*, caching, dan *workload management*. Teknik ini bukan hanya mempercepat sistem, tetapi juga mengurangi pemrosesan yang tidak perlu. FinOps Foundation juga menekankan bahwa optimasi penggunaan sumber daya perlu diselaraskan dengan efisiensi, keberlanjutan, dan nilai bisnis. (FinOps)

Golden Rule: *Data Warehouse* yang ramah lingkungan adalah *Data Warehouse* yang **tidak menyimpan, memproses, memindahkan, dan menghitung data lebih banyak dari yang benar-benar dibutuhkan bisnis**.

14.1 Prinsip *Green Data Warehouse*

Green Data Warehouse adalah pendekatan merancang, membangun, dan mengoperasikan *Data Warehouse* agar penggunaan sumber daya lebih efisien.

Prinsip utamanya:

- **kurangi data yang tidak perlu**;
- **kurangi pemrosesan yang tidak perlu**;
- **kurangi perpindahan data yang tidak perlu**;
- **gunakan storage sesuai nilai data**;
- **jalankan compute sesuai kebutuhan**;
- **optimalkan query dan dashboard**;
- **atur retensi dan arsip data**;
- **ukur biaya dan dampak lingkungan**.

U.S. Department of Energy dan NREL menempatkan pengurangan konsumsi energi sebagai prioritas utama desain pusat data yang efisien, dengan *Power Usage Effectiveness* atau PUE sebagai metrik penting untuk efisiensi energi fasilitas. (The Department of Energy's Energy.gov)

A. Hemat Data

Tidak semua data harus masuk *Data Warehouse*. Tidak semua data harus disimpan selamanya. Tidak semua kolom harus dibawa ke *data mart*. Tidak semua log harus dianalisis pada level detail.

Pertanyaan penting:

- Apakah data ini dipakai untuk keputusan?
- Apakah data ini wajib untuk audit?
- Apakah data ini masih relevan?
- Apakah data ini bisa diarsipkan?
- Apakah data ini mengandung data sensitif?
- Apakah data ini hanya duplikasi dari sumber lain?

Prinsip kritis: data yang tidak dipakai tetap punya biaya: storage, backup, keamanan, retensi, audit, dan risiko kebocoran.

B. Hemat Compute

Compute digunakan untuk *ETL*, transformasi, *query*, agregasi, *dashboard*, validasi kualitas data, dan analitik. Compute yang tidak dikontrol akan memboroskan biaya dan energi.

Contoh pemborosan compute:

- *query* membaca seluruh tabel fakta;
- dashboard menghitung ulang agregasi besar setiap dibuka;
- *ETL* melakukan *full load* padahal cukup *incremental load*;
- data historis lama ikut diproses setiap hari;
- *materialized view* di-*refresh* terlalu sering;
- banyak laporan tidak dipakai tetapi tetap dijalankan.

Optimasi compute:

- gunakan *incremental load*;
- gunakan agregasi;
- gunakan *materialized view* untuk *query* berat yang berulang;
- pisahkan workload ETL dan dashboard;
- jadwalkan proses berat di luar jam sibuk;
- hentikan compute yang tidak dipakai;
- audit dashboard yang jarang dipakai.

Referensi Snowflake yang digunakan menekankan pentingnya memilih ukuran compute sesuai workload, kompleksitas *query*, concurrency, pertumbuhan data, dan biaya. Skala compute harus dapat naik atau turun sesuai kebutuhan agar performa dan biaya tetap seimbang.

C. Hemat Perpindahan Data

Perpindahan data juga memakai energi, bandwidth, storage sementara, dan waktu proses.

Pemborosan umum:

- menarik data penuh setiap hari;
- memindahkan file besar tanpa kompresi;
- menggandakan data antar lingkungan;
- ekspor laporan besar berulang;
- dashboard mengambil data detail dari jarak jauh;
- backup dikirim terlalu sering tanpa kebijakan jelas.

Optimasi:

- gunakan *incremental extraction*;
- gunakan kompresi file;
- minimalkan kolom yang dipindahkan;
- gunakan agregasi dekat sumber jika aman;
- simpan data mart dekat pengguna utama;
- batasi ekspor data besar;
- gunakan retensi file yang jelas.

Referensi yang digunakan menyebut bahwa transfer data yang lebih sedikit dan lebih cepat dapat meningkatkan performa DW melalui kompresi, pengurangan redundansi, dan caching, sehingga latensi dan beban jaringan berkurang.

14.2 Metrik Keberlanjutan untuk DW

Keberlanjutan harus diukur. Tanpa metrik, *green data warehouse* hanya menjadi slogan.

Metrik yang dapat digunakan:

- PUE;
- konsumsi energi;
- storage utilization;
- compute utilization;
- query efficiency;
- data retention efficiency;
- backup efficiency;
- carbon intensity;
- biaya per query;
- biaya per laporan aktif.

A. PUE

PUE atau *Power Usage Effectiveness* adalah metrik efisiensi energi pusat data. The Green Grid menjelaskan PUE sebagai metrik untuk mengukur efisiensi energi infrastruktur pusat data, termasuk pendekatan pengukuran untuk fasilitas khusus maupun campuran. (The Green Grid)

Rumus sederhana:

> PUE = Total energi fasilitas / Energi peralatan IT

Interpretasi:

- semakin mendekati 1, semakin efisien;
- PUE tinggi berarti banyak energi terpakai untuk pendingin, distribusi daya, dan fasilitas pendukung;
- PUE tidak mengukur kualitas aplikasi, tetapi penting untuk memahami efisiensi infrastruktur.

Untuk DW, PUE penting jika organisasi menjalankan platform sendiri. Namun, efisiensi aplikasi tetap harus dikelola melalui optimasi data, query, storage, dan compute.

B. Konsumsi Energi

Konsumsi energi dapat dihitung sederhana:

Energi kWh = daya kW × jam operasi

Contoh:

Server DW = 2 kW
Operasi = 24 jam × 30 hari = 720 jam

Energi = 2 × 720
= 1.440 kWh/bulan

Jika tarif listrik Rp1.500/kWh:

Biaya listrik = 1.440 × Rp1.500
= Rp2.160.000/bulan

Jika memperhitungkan PUE:

Energi fasilitas = energi IT × PUE

Misalnya PUE = 1,6:

Energi fasilitas = 1.440 × 1,6
= 2.304 kWh/bulan

C. *Software Carbon Intensity*

Green Software Foundation menjelaskan *Software Carbon Intensity* atau SCI sebagai metodologi untuk menghitung laju emisi karbon dari sistem perangkat lunak, agar praktisi dapat membuat keputusan desain, pengembangan, dan deployment berbasis bukti untuk menurunkan emisi. (Green Software Foundation) Green Software Foundation juga menyebut SCI sebagai standar ISO/IEC 21031:2024 untuk mengukur emisi karbon aplikasi perangkat lunak. (Green Software Foundation)

Rumus konseptual SCI:

SCI = emisi karbon per unit fungsi

Untuk DW, unit fungsi dapat berupa:

- emisi per 1.000 query;
- emisi per laporan aktif;
- emisi per batch ETL;
- emisi per TB data diproses;
- emisi per dashboard view.

Contoh metrik praktis:

Metrik	Tujuan
kWh per 1 juta query	efisiensi query
kWh per batch ETL	efisiensi pipeline
GB diproses per laporan	efisiensi dashboard
biaya per query	efisiensi finansial
emisi per TB diproses	efisiensi lingkungan

D. *Storage Utilization*

Storage utilization mengukur apakah storage dipakai dengan sehat.

Yang dipantau:

- data aktif;
- raw data;
- staging;
- indeks;
- agregasi;
- backup;
- archive;
- log;
- temporary file.

Contoh rasio:

> Rasio Backup = ukuran backup / ukuran warehouse aktif
>
> Rasio Staging = ukuran staging / ukuran warehouse aktif

Jika staging terlalu besar, mungkin ada file lama yang tidak dibersihkan. Jika backup terlalu besar, mungkin retensi terlalu panjang atau salinan terlalu banyak.

E. *Query Efficiency*

Query efficiency mengukur seberapa hemat *query* menjawab pertanyaan bisnis.

Indikator:

- jumlah data yang dipindai;
- durasi query;
- CPU time;
- memory usage;
- I/O;
- jumlah join;
- apakah memakai partisi;
- apakah memakai agregasi;
- apakah hasil query dipakai.

PostgreSQL menyediakan `EXPLAIN` dan `EXPLAIN ANALYZE` untuk melihat rencana eksekusi query dan mengukur eksekusi nyata. Ini penting untuk menemukan *query* boros. (The Department of Energy's Energy.gov)

14.3 Strategi Desain DW yang Ramah Lingkungan

Strategi desain DW ramah lingkungan tidak harus rumit. Kuncinya adalah mengurangi pemborosan tanpa mengorbankan kualitas data, keamanan, dan kebutuhan bisnis.

Strategi utama:

- desain *grain* yang tepat;
- gunakan *incremental load*;
- gunakan partisi;
- gunakan indeks secukupnya;
- gunakan agregasi dan *materialized view*;
- arsipkan data lama;
- bersihkan staging;
- kurangi dashboard tidak aktif;
- batasi query liar;
- monitor biaya dan energi.

A. Pilih *Grain* yang Tepat

Grain terlalu detail membuat data sangat besar. *Grain* terlalu kasar membuat analisis tidak cukup. Pilih *grain* yang sesuai kebutuhan bisnis.

Contoh:

Kebutuhan	Grain
analisis produk per transaksi	1 baris = 1 item transaksi
analisis invoice	1 baris = 1 invoice
progres proyek	1 baris = 1 progres per minggu
energi fasilitas	1 baris = 1 jam pemakaian
laporan strategis bulanan	agregasi bulanan

Prinsip kritis: simpan detail jika memang dibutuhkan. Jika tidak, gunakan agregasi atau arsip.

B. Gunakan *Incremental Load*

Full load harian untuk tabel besar sangat boros. Gunakan *incremental load* jika sumber data mendukung.

Perbandingan:

Metode	Dampak
Full load	sederhana, tetapi berat
Incremental load	lebih hemat compute dan jaringan
CDC	cocok untuk perubahan cepat, tetapi lebih kompleks
Batch harian	stabil untuk laporan periodik
Near-real-time	mahal jika tidak benar-benar dibutuhkan

Prinsip kritis: pilih frekuensi refresh sesuai keputusan bisnis, bukan sekadar karena teknologi mampu.

C. Partisi Data Historis

Partisi mengurangi jumlah data yang dipindai oleh *query*. Untuk DW, partisi berbasis waktu sering paling alami.

Contoh:

```
fact_sales_2026_01
fact_sales_2026_02
fact_sales_2026_03
```

Manfaat:

- query periode tertentu lebih cepat;
- data lama mudah diarsipkan;
- load per bulan lebih terkontrol;
- maintenance lebih mudah;
- retensi lebih disiplin.

Referensi yang digunakan menjelaskan bahwa *partitioning* membagi tabel besar menjadi bagian lebih kecil agar akses data dan pemrosesan dapat lebih efisien.

D. Gunakan Agregasi dan *Materialized View*

Dashboard tidak seharusnya selalu membaca data detail. Untuk laporan berulang, gunakan agregasi atau *materialized view*.

Contoh:

```
fact_sales_detail
    ↓
mart_sales_daily
    ↓
mart_sales_monthly
    ↓
dashboard_executive
```

Manfaat:

- query lebih cepat;
- compute lebih hemat;
- dashboard lebih stabil;
- beban tabel fakta menurun.

Referensi menyebut *materialized views* sebagai hasil query kompleks yang sudah dihitung sebelumnya sehingga mengurangi kebutuhan menghitung ulang hasil yang sama berulang kali.

E. Terapkan Retensi dan Arsip

Retensi menentukan berapa lama data disimpan aktif. Arsip memindahkan data lama ke storage yang lebih murah atau lebih jarang diakses.

Contoh kebijakan:

Jenis Data	Aktif	Arsip	Catatan
transaksi detail	2 tahun	5 tahun	untuk audit
data agregat bulanan	5 tahun	10 tahun	untuk tren
staging raw	3 bulan	sesuai kebutuhan	untuk troubleshooting
log ETL	1 tahun	3 tahun	untuk audit proses
dashboard tidak aktif	6 bulan	hapus / arsip	sesuai persetujuan

Referensi *Mastering Data Warehousing* menjelaskan bahwa *Information Lifecycle Management* mengelola data dari penciptaan sampai pembuangan, termasuk performa, redundansi, keamanan, retensi, dan arsip.

14.4 Optimasi Operasional Ramah Lingkungan

Setelah produksi, DW harus dikelola agar tidak boros.

Aktivitas operasional:

- mematikan proses yang tidak dipakai;
- jadwalkan ETL di luar jam sibuk;
- audit dashboard tidak aktif;
- bersihkan staging;
- arsipkan data lama;
- optimasi query lambat;
- review indeks;
- review backup;
- pantau storage growth;
- pantau compute utilization.

A. Audit Dashboard

Dashboard yang tidak dipakai tetap membutuhkan biaya.

Yang dicek:

- dashboard mana yang tidak dibuka 3 bulan;
- dashboard mana yang terlalu lambat;
- dashboard mana yang membaca data detail langsung;
- dashboard mana yang duplikat;
- laporan mana yang bisa digabung;
- laporan mana yang bisa diganti agregasi.

Contoh kebijakan:

Kondisi	**Tindakan**
tidak dipakai 3 bulan	review dengan owner
tidak dipakai 6 bulan	arsip
duplikat	gabungkan
query terlalu berat	optimasi
data sensitif tidak perlu	masking / hapus kolom

B. Review *ETL Window*

ETL harus dijadwalkan agar hemat dan tidak mengganggu pengguna.

Checklist:

- apakah ETL berjalan saat dashboard ramai?
- apakah ETL masih memakai *full load*?
- apakah proses berat bisa dipisah?
- apakah validasi bisa dijalankan bertahap?
- apakah data historis lama ikut diproses terus?
- apakah error membuat proses mengulang dari nol?

C. Review Backup

Backup harus aman, tetapi tidak boleh boros tanpa alasan.

Yang dicek:

- jumlah salinan;
- retensi;
- backup data aktif;
- backup arsip;
- backup metadata;
- backup script ETL;
- ukuran backup;
- restore test;
- enkripsi backup.

Prinsip kritis: backup harus cukup untuk pemulihan, bukan tak terbatas tanpa kebijakan.

D. Review Query Berat

Query berat harus dianalisis karena berdampak pada biaya, energi, dan pengalaman pengguna.

Langkah:

1. identifikasi query paling mahal;
2. cek apakah query dipakai;
3. cek apakah bisa memakai agregasi;
4. cek indeks dan partisi;
5. cek jumlah kolom yang dibaca;
6. cek apakah ada filter waktu;
7. ubah menjadi *materialized view* jika sering dipakai;
8. batasi query eksploratif di jam sibuk.

14.5 Rumus Dasar Efisiensi DW

Gunakan rumus sederhana untuk latihan sertifikasi.

A. Efisiensi Storage

Storage Efficiency Ratio = Data Aktif yang Dipakai / Total Storage

Contoh:

Data aktif dipakai = 10 TB
Total storage = 60 TB

Storage Efficiency Ratio = 10 / 60
= 16,7%

Jika rasio terlalu rendah, cek backup, staging, log, archive, dan data tidak aktif.

B. Efisiensi Query

Biaya per Query = Total biaya bulanan / Jumlah query bulanan
Energi per Query = Total energi bulanan / Jumlah query bulanan

Contoh:

Energi bulanan = 2.304 kWh
Query bulanan = 1.000.000

Energi per query = 2.304 / 1.000.000
= 0,002304 kWh/query

C. Efisiensi Dashboard

Biaya per Dashboard Aktif = Total biaya dashboard / Jumlah dashboard aktif
Usage Ratio = Dashboard aktif / Total dashboard

Contoh:

Total dashboard = 80
Dashboard aktif = 40

Usage Ratio = 40 / 80
= 50%

Artinya, separuh dashboard tidak memberi nilai aktif tetapi mungkin masih menimbulkan biaya.

D. Efisiensi Retensi

Rasio Data Lama = Data melewati periode aktif / Total data

Jika rasio data lama terlalu besar, buat kebijakan arsip.

Hands-on. Membuat Checklist Environmentally Friendly Data Warehouse

Gunakan template ini untuk menilai apakah DW sudah efisien dan ramah lingkungan.

A. Identitas Sistem

Elemen	Isi
Nama DW	
Domain data	
Pemilik bisnis	
Pemilik teknis	
Total storage	
Jumlah dashboard	
Jumlah user	
Refresh utama	
Retensi utama	

B. Checklist Data dan Storage

Pertanyaan	Ya/Tidak	Catatan
Apakah semua dataset punya pemilik?		
Apakah semua dataset punya retensi?		
Apakah staging dibersihkan berkala?		
Apakah data lama diarsipkan?		
Apakah backup punya kebijakan retensi?		
Apakah log punya batas retensi?		
Apakah data duplikat dikurangi?		
Apakah kolom tidak dipakai dibuang dari data mart?		

C. Checklist Compute dan Query

Pertanyaan	Ya/Tidak	Catatan
Apakah ETL memakai incremental load jika memungkinkan?		
Apakah dashboard membaca data agregat?		
Apakah query berat dimonitor?		
Apakah materialized view dipakai untuk laporan berulang?		
Apakah query tanpa filter waktu dibatasi?		
Apakah proses berat dijadwalkan di luar jam sibuk?		
Apakah workload analis dipisah dari dashboard utama?		
Apakah compute yang tidak dipakai dihentikan / dikurangi?		

D. Checklist Dashboard dan Laporan

Pertanyaan	Ya/Tidak	Catatan
Apakah dashboard tidak aktif di review?		
Apakah dashboard duplikat digabung?		
Apakah dashboard lambat dioptimasi?		
Apakah laporan tidak dipakai diarsipkan?		
Apakah dashboard eksekutif memakai data ringkasan?		
Apakah ekspor data besar dibatasi?		

E. Checklist Keberlanjutan dan Biaya

Metrik	Nilai	Catatan
biaya per query		
biaya per dashboard aktif		
storage aktif		
storage backup		
rasio staging terhadap warehouse		
query bulanan		
dashboard aktif		
konsumsi energi / bulan		
estimasi energi per query		

F. Template Rencana Optimasi

Masalah	Dampak	Tindakan	Prioritas
staging terlalu besar	storage boros	bersihkan sesuai retensi	Tinggi
dashboard tidak dipakai	compute boros	arsip setelah review	Sedang
query lambat	compute tinggi	indeks / agregasi	Tinggi
full load harian	compute dan jaringan boros	ubah ke incremental	Tinggi
backup terlalu banyak	storage boros	review retensi	Sedang
data lama aktif	query lambat	archive	Tinggi

Contoh Mini Audit *Green DW*

Kasus:

- total storage = 60 TB;
- data aktif = 10 TB;
- backup = 25 TB;
- staging = 8 TB;
- archive = 12 TB;
- log = 5 TB;
- dashboard total = 80;
- dashboard aktif = 40;
- query bulanan = 1 juta;
- energi bulanan = 2.304 kWh.

A. Rasio Storage Aktif

Data aktif / total storage = 10 / 60 = 16,7%

Interpretasi: perlu cek apakah backup, staging, log, dan arsip terlalu besar.

B. Rasio Dashboard Aktif

Dashboard aktif / total dashboard = 40 / 80 = 50%

Interpretasi: 50% dashboard perlu direview, diarsipkan, atau digabung.

C. Energi per Query

Energi per query = 2.304 kWh / 1.000.000
= 0,002304 kWh/query

Gunakan metrik ini sebagai baseline. Setelah optimasi query dan dashboard, angka ini harus turun.

Kesalahan Umum

Kesalahan yang sering terjadi:

- semua data dimasukkan ke DW tanpa seleksi;
- semua data disimpan selamanya;
- dashboard membaca tabel detail langsung;
- *full load* dilakukan harian tanpa alasan;
- staging tidak pernah dibersihkan;
- backup dibuat banyak tetapi tanpa retensi jelas;
- query lambat dibiarkan;
- data historis lama tetap aktif;
- laporan tidak dipakai tetap di-*refresh*;
- tidak menghitung biaya per query;
- tidak menghitung energi per workload;
- optimasi biaya dianggap hanya tugas keuangan;
- keberlanjutan dianggap terpisah dari arsitektur data.

Prinsip kritis: DW yang boros biasanya bukan karena datanya besar, tetapi karena **desain, retensi, query, dan operasi tidak disiplin**.

Ringkasan Bab

Environmentally friendly Data Warehouse adalah pendekatan untuk membuat DW lebih hemat energi, hemat biaya, dan lebih mudah dikendalikan. Strateginya bukan sekadar memilih infrastruktur efisien, tetapi mengurangi pemborosan data, compute, storage, backup, jaringan, dashboard, dan query. Metrik seperti PUE, konsumsi energi, biaya per query, storage utilization, dashboard usage, dan SCI membantu organisasi mengukur efisiensi secara lebih objektif.

Poin penting:

- **Data yang tidak dipakai tetap punya biaya dan risiko.**
- *Incremental load* lebih hemat daripada *full load* berulang.
- Partisi, indeks, agregasi, caching, dan *materialized view* dapat mengurangi pemrosesan berulang.
- Retensi dan arsip adalah bagian penting dari sustainability.
- Dashboard yang tidak dipakai harus direview.
- Backup harus cukup, aman, dan tidak berlebihan.
- FinOps dan sustainability saling mendukung melalui optimasi penggunaan sumber daya.
- Efisiensi lingkungan harus dikaitkan dengan nilai bisnis, bukan hanya penghematan teknis.

Latihan Sertifikasi

A. Pertanyaan Konseptual

1. Mengapa *Data Warehouse* dapat menjadi boros energi dan biaya?
2. Apa hubungan antara optimasi query dan keberlanjutan?
3. Jelaskan fungsi PUE dalam konteks pusat data.
4. Mengapa data retensi penting untuk *green data warehouse*?
5. Apa risiko dashboard yang tidak dipakai tetapi tetap aktif?
6. Mengapa *incremental load* lebih ramah lingkungan daripada *full load* harian?
7. Apa fungsi *Software Carbon Intensity* dalam pengukuran dampak perangkat lunak?
8. Mengapa optimasi biaya dan sustainability harus dikerjakan lintas peran?

B. Studi Kasus

Sebuah DW memiliki storage total 60 TB, tetapi data aktif hanya 10 TB. Dashboard total 80, tetapi yang aktif hanya 40. ETL harian masih memakai *full load*, dan query dashboard sering membaca tabel fakta detail.

Tugas:

- hitung rasio data aktif terhadap total storage;
- hitung rasio dashboard aktif;
- identifikasi 5 pemborosan utama;
- buat 5 rekomendasi optimasi;
- tentukan metrik monitoring sustainability;
- susun rencana 3 bulan untuk mengurangi pemborosan.

C. Praktek

Buat dokumen evaluasi *Environmentally Friendly Data Warehouse* dengan format:

Area	Kondisi Saat Ini	Risiko	Rekomendasi
Storage			
Backup			
Staging			
Query			
Dashboard			
ETL			
Retensi			
Energi			
Biaya			
Sustainability KPI			

BAB 15. Roadmap Implementasi dan Persiapan Sertifikasi *Data Warehouse*

Bab terakhir ini menyatukan seluruh isi buku menjadi dua jalur praktis: **roadmap implementasi 90 hari** dan **roadmap belajar sertifikasi 6 minggu**. Tujuannya agar pembaca tidak hanya memahami konsep, tetapi mampu menyusun langkah nyata untuk membangun, menguji, mengamankan, mengoperasikan, dan mengevaluasi *Data Warehouse*.

Referensi yang digunakan menekankan bahwa roadmap *Data Warehouse* harus dimulai dari tujuan yang jelas, perencanaan menyeluruh, kolaborasi *stakeholder*, implementasi bertahap, pengujian, deployment, monitoring, maintenance, dan evaluasi berkala. DAMA-DMBOK juga menempatkan manajemen data sebagai disiplin standar bagi profesional data agar organisasi dapat membangun praktik pengelolaan data yang konsisten dan efektif. (DAMA International®)

Golden Rule: sertifikasi *Data Warehouse* bukan hanya menghafal istilah, tetapi membuktikan bahwa peserta mampu **menerjemahkan kebutuhan bisnis menjadi arsitektur, model, pipeline, kontrol, biaya, dan operasi yang dapat berjalan**.

15.1 Roadmap Implementasi 90 Hari

Roadmap 90 hari adalah rencana implementasi cepat untuk membangun *Data Warehouse* versi awal yang realistis. Targetnya bukan membangun semua domain data sekaligus, tetapi menghasilkan **MVP yang valid, aman, terdokumentasi, dan siap diperluas**.

Roadmap ini cocok untuk:

- proyek sertifikasi;
- proyek percontohan;
- unit bisnis yang ingin membangun DW awal;
- organisasi yang ingin memulai dari 1–2 domain data;
- tim kecil sampai menengah.

Referensi *Mastering Data Warehousing* menekankan pentingnya *source data assessment*, *data cleanup*, desain dan modeling, pengembangan ETL, performance tuning, testing, security, deployment, training, monitoring, iterative growth, maintenance, backup, recovery, dan review berkelanjutan.

Hari 1–15: *Assessment* Kebutuhan dan Sumber Data

Fokus fase ini adalah memahami **masalah bisnis, keputusan yang ingin didukung, KPI, pengguna, dan sumber data**.

Output utama:

- *project charter*;
- daftar *stakeholder*;
- daftar laporan dan dashboard;
- daftar KPI;
- daftar sumber data;
- analisis kualitas data awal;
- batasan *scope* MVP.

Aktivitas:

- wawancara sponsor bisnis;
- wawancara pengguna utama;
- identifikasi laporan lama;
- identifikasi sistem sumber;
- cek format data;
- cek volume data;
- cek data sensitif;
- cek pemilik data;
- tentukan domain awal.

Checklist:

Pertanyaan	Jawaban
keputusan apa yang ingin didukung?	
siapa pengguna utama?	
KPI apa yang paling penting?	
sumber data apa yang tersedia?	
data mana yang sensitif?	
kualitas data awal seperti apa?	
domain mana yang masuk MVP?	

Prinsip kritis: jangan mulai dari tabel. Mulai dari keputusan.

Hari 16–30: Desain Arsitektur dan Model Data

Fase ini menerjemahkan kebutuhan bisnis menjadi desain teknis.

Output utama:

- arsitektur DW;
- desain *staging*;
- desain *warehouse core*;
- desain *data mart*;
- *star schema*;
- *fact table*;
- *dimension table*;
- *grain*;
- *data dictionary* awal.

Aktivitas:

- gambar alur sumber data ke dashboard;
- tentukan *ETL* atau *ELT*;
- tentukan tabel fakta dan dimensi;
- tentukan *grain*;
- tentukan SCD;
- tentukan metadata minimum;
- tentukan akses awal.

Kimball Group menempatkan *fact table*, *dimension table*, *conformed dimensions*, dan *slowly changing dimensions* sebagai teknik inti *dimensional modeling*. (Kimball Group)

Contoh output model:

```
Source Data → Raw Staging → Clean Staging → FactSales + Dimensi → Data Mart → Dashboard
```

Hari 31–50: *ETL/ELT* dan *Staging*

Fase ini membangun aliran data.

Output utama:

- tabel raw;
- tabel staging;
- script extract;
- script transform;
- script load;
- tabel error;
- tabel log;
- *source-to-target mapping*;
- rekonsiliasi awal.

Aktivitas:

- load file / database / API ke raw;
- bersihkan data;
- standarisasi kode;
- deduplikasi;
- validasi field wajib;
- load dimensi;
- load fakta;
- catat row count;
- catat error count;
- buat proses ulang yang aman.

Referensi ETL menjelaskan bahwa proses ETL mencakup ekstraksi data dari berbagai sumber, transformasi menjadi format konsisten, dan pemuatan ke DW atau *data mart*.

Hari 51–65: *Data Mart* dan *Dashboard*

Fase ini mengubah DW menjadi produk informasi yang dapat dipakai.

Output utama:

- *data mart*;
- view ringkasan;
- query KPI;
- dashboard awal;
- filter utama;
- validasi angka;
- dokumentasi KPI.

Aktivitas:

- buat mart penjualan / proyek / keuangan;
- buat agregasi bulanan;
- buat query dashboard;
- validasi dengan laporan lama;
- uji performa dashboard;
- minta masukan pengguna.

Contoh KPI:

- total penjualan;
- margin;
- progres proyek;
- deviasi biaya;
- jumlah pelanggan aktif;
- SLA;
- biaya per unit.

Hari 66–75: *Security*, *Governance*, dan Testing

Fase ini memastikan sistem aman, legal, dan dapat dipercaya.

Output utama:

- klasifikasi data;
- *access matrix*;
- role pengguna;
- masking;
- audit log;
- test case;
- data quality report;
- security checklist.

Aktivitas:

- klasifikasi Public, Internal, Confidential, Restricted, Personal Data;
- buat role Executive, Analyst, Engineer, Auditor, Admin;
- uji akses;
- uji masking;
- uji kualitas data;
- uji duplikasi;
- uji null value;
- uji rekonsiliasi;
- uji performa dasar.

Metadata mendukung governance, compliance, kualitas data, dokumentasi, integrasi, ownership, pembatasan akses, dan pola penggunaan data.

Hari 76–85: UAT dan Deployment

Fase ini membuktikan sistem layak dipakai pengguna.

Output utama:

- UAT plan;
- hasil UAT;
- daftar isu;
- perbaikan;
- sign-off bisnis;
- deployment checklist;
- backup awal;
- monitoring awal;
- rollback plan.

Aktivitas:

- jalankan skenario UAT;
- validasi KPI;
- validasi filter;
- validasi akses;
- validasi dashboard;
- siapkan backup;
- siapkan monitoring;
- siapkan *cutover*;
- siapkan rollback.

Prinsip kritis: UAT bukan demo. UAT adalah pembuktian bahwa sistem dapat menjadi rujukan keputusan.

Hari 86–90: Dokumentasi, Training, dan Review

Fase terakhir memastikan sistem tidak hanya selesai, tetapi dapat dirawat.

Output utama:

- *data dictionary*;
- *ETL mapping*;
- *data lineage*;
- *operational manual*;
- training pengguna;
- training tim operasi;
- review hasil MVP;
- roadmap fase berikutnya.

Aktivitas:

- finalisasi dokumentasi;
- pelatihan dashboard;
- pelatihan operasi harian;
- review biaya;
- review performa;
- review kualitas data;
- susun backlog fase berikutnya.

Referensi roadmap DW menekankan bahwa setelah deployment, DW harus terus berkembang melalui feedback pengguna, scaling, maintenance, backup, recovery, tuning, dan review kesesuaian dengan tujuan bisnis.

15.2 Roadmap Belajar Sertifikasi

Roadmap belajar 6 minggu dirancang agar peserta memahami teori dan mampu mengerjakan studi kasus.

Minggu 1: Konsep DW, OLTP, OLAP

Target kompetensi:

- menjelaskan definisi DW;
- membedakan OLTP dan OLAP;
- menjelaskan fungsi DW untuk keputusan;
- memahami *subject-oriented*, *integrated*, *time-variant*, *non-volatile*;
- memahami data historis dan analitik.

Materi:

- konsep dasar DW;
- OLTP vs OLAP;
- manfaat DW;
- contoh kasus bisnis;
- peran DW dalam audit dan analitik.

Latihan:

- buat peta sumber data organisasi;
- jelaskan kebutuhan laporan strategis;
- identifikasi KPI utama.

Minggu 2: Arsitektur dan *ETL/ELT*

Target kompetensi:

- menggambar arsitektur DW;
- menjelaskan *source system*, staging, DW core, data mart, dashboard;
- membedakan ETL dan ELT;
- menjelaskan extract, transform, load;
- memahami logging dan audit ETL.

Materi:

- arsitektur tiga lapis;
- staging;
- ETL vs ELT;
- full extract;
- incremental extract;
- CDC;
- file ingestion;
- API ingestion;
- error handling.

Latihan:

- gambar arsitektur DW sederhana;
- buat alur CSV → staging → fact table;
- buat tabel log ETL.

Minggu 3: *Dimensional Modeling*

Target kompetensi:

- menentukan *grain*;
- membuat *fact table*;
- membuat *dimension table*;
- menjelaskan *measure*;
- membuat *star schema*;
- memahami *snowflake schema*;
- memilih SCD Type 0–3.

Materi:

- fact;
- dimension;
- measure;
- grain;
- hierarchy;
- surrogate key;
- star schema;
- snowflake schema;
- SCD.

Latihan:

- buat *star schema* penjualan;
- tentukan grain;
- tentukan dimensi dan measure;
- pilih SCD untuk pelanggan dan produk.

Minggu 4: *Security*, *Governance*, Metadata

Target kompetensi:

- membuat klasifikasi data;
- membuat *access matrix*;
- menjelaskan RBAC;
- memahami masking;
- memahami audit log;
- membuat *data dictionary*;
- membuat *data lineage*.

Materi:

- *least privilege*;
- *segregation of duties*;
- klasifikasi data;
- personal data;
- metadata;
- data quality;
- compliance;
- audit readiness.

Latihan:

- buat matriks akses;
- klasifikasikan dataset;
- buat data lineage dari sumber sampai dashboard.

Minggu 5: *Performance Tuning*

Target kompetensi:

- menjelaskan faktor performa;
- memahami indeks;
- memahami partisi;
- memahami agregasi;
- memahami *materialized view*;
- membaca baseline monitoring;
- menghitung kapasitas awal.

Materi:

- volume data;
- jumlah user;
- kompleksitas query;
- refresh frequency;
- indexing;
- partitioning;
- query response time;
- CPU, memory, I/O;
- ETL duration.

Latihan:

- hitung ukuran fact table;
- hitung storage 5 tahun;
- usulkan indeks dan partisi;
- buat baseline monitoring.

Minggu 6: Cost, SDM, Deployment, Operasi

Target kompetensi:

- menghitung TCO;
- menghitung biaya SDM;
- membuat deployment checklist;
- membuat operasi harian;
- memahami backup dan DR;
- memahami sustainability.

Materi:

- CAPEX vs OPEX;
- TCO 3 tahun;
- biaya cloud / on-premise;
- struktur tim;
- deployment;
- backup;
- RPO / RTO;
- FinOps;
- *green data warehouse*.

FinOps Foundation menekankan optimasi biaya sebagai proses merancang, membangun, dan mengoperasikan sistem secara efektif, mengurangi pemborosan, dan memastikan biaya selaras dengan nilai bisnis. (FinOps)

Latihan:

- hitung biaya DW 3 tahun;
- buat struktur tim;
- buat checklist deployment;
- buat checklist operasi harian.

15.3 Kisi-Kisi Ujian

Kisi-kisi ini menyatukan kompetensi dari seluruh buku. Ujian sertifikasi sebaiknya tidak hanya menguji hafalan, tetapi juga kemampuan desain dan analisis kasus.

A. Topik Ujian

Topik	Kompetensi yang Diuji
Konsep dasar DW	definisi, tujuan, manfaat, karakteristik
Arsitektur	source, staging, DW core, data mart, dashboard
ETL/ELT	extract, transform, load, logging, audit
Data modeling	fact, dimension, grain, star, snowflake, SCD
OLAP	slice, dice, drill-down, roll-up, pivot
Metadata	data dictionary, lineage, mapping, ownership
Data quality	completeness, accuracy, consistency, timeliness, uniqueness, validity
Security	RBAC, masking, audit log, klasifikasi data
Deployment	environment, UAT, cutover, rollback
Performance	index, partition, aggregation, monitoring
Cost planning	TCO, CAPEX, OPEX, biaya per query
SDM	peran tim, RACI, kompetensi
Sustainability	efisiensi storage, compute, query, retensi

B. Bentuk Soal

Ujian dapat memakai kombinasi:

- pilihan ganda;
- benar / salah;
- mencocokkan istilah;
- studi kasus pendek;
- studi kasus panjang;
- desain skema;
- perhitungan kapasitas;
- perhitungan biaya;
- analisis risiko;
- checklist deployment.

Contoh bobot:

Bentuk Ujian	**Bobot**
Konsep dasar	20%
Desain arsitektur dan modeling	25%
ETL/ELT dan kualitas data	20%
Security, deployment, operasi	20%
Cost, SDM, sustainability	15%

C. Level Kompetensi

Level	**Deskripsi**
Level 1	memahami istilah dan konsep
Level 2	menjelaskan fungsi dan perbedaan
Level 3	menerapkan pada kasus sederhana
Level 4	menganalisis risiko dan trade-off
Level 5	menyusun desain lengkap dan mempertahankan keputusan

Untuk target profesional, ujian idealnya dominan Level 3–4, dengan beberapa soal Level 5 berbasis studi kasus.

15.4 Simulasi Studi Kasus

Simulasi studi kasus adalah bagian terpenting dari persiapan sertifikasi. Peserta harus mampu menyusun solusi lengkap, bukan hanya menjawab definisi.

Kasus

Sebuah organisasi ingin membangun *Data Warehouse* untuk domain penjualan dan proyek. Data berasal dari:

- transaksi penjualan;
- pelanggan;
- produk;
- cabang;
- invoice;
- progres proyek mingguan;
- biaya proyek;
- vendor;
- target bulanan.

Pengguna:

- direksi;
- manajer unit;
- analis;
- auditor;
- tim data.

Kebutuhan:

- dashboard penjualan;
- dashboard margin;
- dashboard progres proyek;
- dashboard deviasi biaya;
- laporan audit;
- refresh harian untuk penjualan;
- refresh mingguan untuk proyek;
- retensi 5 tahun;
- data sensitif harus dimasking;
- deployment 90 hari.

Tugas 1 — Mendesain Arsitektur

Peserta diminta menggambar arsitektur:

```
Source System
    ↓
Raw Staging
    ↓
Clean Staging
    ↓
Data Warehouse Core
    ↓
Data Mart
    ↓
Dashboard / OLAP / Audit Report
```

Wajib menjelaskan:

- sumber data;
- staging;
- ETL/ELT;
- DW core;
- data mart;
- metadata;
- security;
- monitoring;
- backup.

Tugas 2 — Membuat *Star Schema*

Minimal buat dua model:

Model Penjualan

Komponen	Isi
Fact	FactSales
Grain	1 baris = 1 item transaksi penjualan
Dimensi	DimCustomer, DimProduct, DimDate, DimBranch, DimSalesChannel
Measure	quantity, gross_amount, discount, net_amount, cost_amount, margin_amount

Model Proyek

Komponen	Isi
Fact	FactProjectProgress
Grain	1 baris = 1 progres proyek per minggu
Dimensi	DimProject, DimDate, DimVendor, DimLocation, DimWorkPackage
Measure	planned_progress, actual_progress, progress_variance, planned_cost, actual_cost, cost_variance

Tugas 3 — Menentukan *ETL/ELT*

Peserta harus menjelaskan:

- metode extract;
- sumber full extract;
- sumber incremental extract;
- transformasi;
- cleansing;
- deduplikasi;
- masking;
- load dimensi;
- load fakta;
- logging;
- rekonsiliasi.

Contoh:

Sumber	**Metode**	**Catatan**
transaksi penjualan	incremental	berdasarkan updated_at
pelanggan	full / incremental	tergantung volume
produk	full mingguan	master data kecil
progres proyek	file mingguan	validasi manual
invoice	incremental	audit penting

Tugas 4 — Menghitung Kapasitas

Gunakan rumus:

> Ukuran Fact = jumlah baris × ukuran rata-rata baris
> Storage Total = Raw + Staging + Warehouse + Index + Backup + Retention

Peserta harus menghitung:

- ukuran data bulanan;
- ukuran data tahunan;
- pertumbuhan 5 tahun;
- index overhead;
- backup;
- staging;
- cadangan;
- rekomendasi kapasitas.

Tugas 5 — Menghitung Biaya

Gunakan rumus:

> TCO 3 Tahun = biaya implementasi awal + total biaya operasional 36 bulan

Peserta harus menghitung:

- biaya implementasi;
- biaya storage;
- biaya compute;
- biaya backup;
- biaya monitoring;
- biaya SDM;
- biaya training;
- biaya security;
- cadangan risiko;
- biaya per query;
- biaya per laporan.

Tugas 6 — Menentukan Tim

Peserta harus menyusun struktur tim:

Peran	Kebutuhan
Sponsor bisnis	wajib
Product owner data	wajib
Data architect	wajib
Data engineer	wajib
DBA / platform engineer	wajib
BI developer	wajib
Data steward	wajib
Security reviewer	wajib
PM	wajib

Peserta juga harus menentukan peran yang boleh dirangkap dan risiko perangkapan tersebut.

Tugas 7 — Membuat Checklist Deployment

Checklist minimum:

Area	Item
Environment	development, testing, staging, production
Database	skema, tabel, indeks, constraint
ETL	extract, transform, load, log
Data Quality	null, duplicate, referential integrity
Security	role, masking, audit log
Backup	full backup, restore test
UAT	skenario, hasil, sign-off
Monitoring	ETL, query, storage, backup
Rollback	snapshot, restore procedure, kontak darurat

Hands-on. Template Rencana Sertifikasi dan Implementasi

Gunakan template berikut sebagai latihan akhir.

A. Roadmap Implementasi 90 Hari

Periode	Aktivitas	Output
Hari 1–15	assessment kebutuhan dan sumber data	charter, KPI, sumber data
Hari 16–30	desain arsitektur dan model data	arsitektur, star schema
Hari 31–50	ETL/ELT dan staging	pipeline, staging, log
Hari 51–65	data mart dan dashboard	dashboard awal
Hari 66–75	security, governance, testing	access matrix, test report
Hari 76–85	UAT dan deployment	sign-off, deployment checklist
Hari 86–90	dokumentasi, training, review	manual, training, backlog

B. Roadmap Belajar 6 Minggu

Minggu	Fokus	Output
1	konsep DW, OLTP, OLAP	ringkasan konsep
2	arsitektur dan ETL/ELT	diagram arsitektur
3	dimensional modeling	star schema
4	security, governance, metadata	access matrix, lineage
5	performance tuning	kapasitas dan tuning plan
6	cost, SDM, deployment, operasi	TCO, tim, checklist

C. Template Jawaban Studi Kasus

Komponen	Jawaban
Tujuan bisnis	
Pengguna utama	
KPI utama	
Sumber data	
Arsitektur	
Fact table	
Dimension table	
Grain	
ETL/ELT	
Data quality	
Security	
Kapasitas	
Biaya	
Tim	
Deployment	
Operasi	
Risiko	
Mitigasi	

Checklist Kesiapan Sertifikasi

Kompetensi	Siap?
Menjelaskan konsep DW, OLTP, OLAP	
Menggambar arsitektur DW	
Menentukan ETL atau ELT	
Membuat star schema	
Menentukan grain	
Menentukan SCD	
Membuat metadata dan lineage	
Membuat aturan kualitas data	
Membuat klasifikasi data	
Membuat access matrix	
Menghitung kapasitas storage	
Menentukan indeks dan partisi	
Menghitung TCO	
Menentukan struktur tim	
Membuat deployment checklist	
Membuat checklist operasi harian	
Menjelaskan sustainability DW	

Kesalahan Umum Saat Sertifikasi

Kesalahan yang sering terjadi:

- menghafal istilah tanpa bisa menerapkan;
- membuat arsitektur tanpa kebutuhan bisnis;
- membuat fact table tanpa grain;
- lupa metadata dan lineage;
- mengabaikan security;
- tidak menghitung backup dalam kapasitas;
- tidak menghitung SDM dalam biaya;
- tidak membuat rollback plan;
- tidak membedakan UAT dan testing teknis;
- tidak membuat checklist operasi;
- tidak bisa menjelaskan trade-off antara biaya, performa, dan keamanan.

Prinsip kritis: jawaban sertifikasi yang baik harus menunjukkan **alasan desain**, bukan hanya menyebut komponen.

Ringkasan Bab

Bab ini menutup buku dengan roadmap implementasi dan roadmap belajar. Roadmap 90 hari membantu organisasi membangun DW awal secara bertahap: assessment, desain, ETL, data mart, security, UAT, deployment, dokumentasi, dan review. Roadmap belajar 6 minggu membantu peserta sertifikasi menguasai konsep, arsitektur, ETL/ELT, modeling, security, governance, performance, biaya, SDM, deployment, operasi, dan sustainability.

Poin penting:

- **Mulai dari kebutuhan bisnis dan sumber data.**
- Bangun MVP kecil tetapi benar.
- Gunakan *star schema* untuk model awal yang mudah dipahami.
- ETL/ELT harus memiliki log, error handling, dan rekonsiliasi.
- Security, governance, dan testing harus hadir sebelum deployment.
- UAT harus menghasilkan *business sign-off*.
- Sertifikasi harus menguji kemampuan desain, bukan sekadar hafalan.
- Studi kasus adalah cara terbaik membuktikan kompetensi profesional.
- Roadmap implementasi dan roadmap belajar harus berakhir pada kemampuan nyata: **mendesain, membangun, mengamankan, menghitung, mengoperasikan, dan mengevaluasi DW**.

Latihan Sertifikasi Akhir

A. Pertanyaan Konseptual

1. Mengapa roadmap implementasi DW sebaiknya dimulai dari assessment kebutuhan bisnis?
2. Apa output utama dari fase desain arsitektur dan model data?
3. Mengapa UAT harus dilakukan sebelum deployment?
4. Apa perbedaan roadmap implementasi dan roadmap belajar sertifikasi?
5. Sebutkan minimal 10 topik penting dalam kisi-kisi sertifikasi DW.
6. Mengapa studi kasus lebih baik daripada soal hafalan untuk sertifikasi profesional?
7. Apa risiko jika peserta mampu membuat dashboard tetapi tidak memahami data lineage?
8. Bagaimana FinOps dan sustainability masuk ke kompetensi sertifikasi DW?

B. Studi Kasus Akhir

Sebuah organisasi ingin membangun DW dalam 90 hari untuk domain penjualan dan proyek. Data berasal dari transaksi, pelanggan, produk, cabang, proyek, vendor, progres, dan biaya. Pengguna terdiri dari direksi, manajer, analis, auditor, dan tim data.

Tugas:

- buat roadmap 90 hari;
- gambar arsitektur;
- buat *star schema* penjualan;
- buat *star schema* proyek;
- tentukan ETL/ELT;
- tentukan aturan kualitas data;
- buat access matrix;
- hitung kapasitas dasar;
- hitung biaya 3 tahun;
- tentukan struktur tim;
- buat checklist deployment;
- buat checklist operasi harian;
- tentukan 5 risiko utama dan mitigasinya.

Lampiran A. Template Data Warehouse Project Charter

A.1 Identitas Proyek

Komponen	Isi
Nama proyek	
Versi dokumen	
Tanggal	
Unit pengusul	
Sponsor bisnis	
Project manager	
Product owner data	

A.2 Tujuan Bisnis

Tuliskan masalah bisnis yang ingin diselesaikan dan keputusan apa yang ingin didukung.

Pertanyaan	Jawaban
Masalah utama	
Keputusan bisnis yang didukung	
Manfaat yang diharapkan	
Prioritas bisnis	Tinggi / Sedang / Rendah

Contoh tujuan bisnis:

- Menyediakan **satu sumber data resmi** untuk laporan penjualan, proyek, biaya, dan KPI.
- Mengurangi proses laporan manual.
- Mempercepat akses data historis untuk analisis dan audit.
- Meningkatkan konsistensi definisi KPI antar unit.

A.3 *Scope* Proyek

Area	Masuk Scope	Tidak Masuk Scope
Domain data		
Periode data historis		
Jenis laporan		
Dashboard		
Integrasi sistem		
Keamanan dan akses		

Catatan penting:

- **Scope awal sebaiknya kecil tetapi bernilai tinggi.**
- Hindari membangun semua domain sekaligus.
- Gunakan pendekatan **MVP → pilot → production → perluasan**.

A.4 *Data Source*

No.	Sumber Data	Pemilik Data	Format	Frekuensi Update	Risiko Kualitas
1			Database / CSV / API / Log	Harian / Mingguan / Bulanan	
2					
3					

Contoh risiko kualitas:

- data kosong;
- duplikasi;
- format tanggal tidak konsisten;
- kode cabang tidak valid;
- definisi KPI berbeda antar unit.

A.5 Target Pengguna

Kelompok Pengguna	Kebutuhan Utama	Jenis Akses
Direksi	Dashboard ringkasan KPI	Read-only
Manajer unit	Laporan unit / wilayah	Read-only terbatas
Analis	Analisis detail	Read terbatas
Auditor	Data, lineage, dan log	Read-only sesuai mandat
Tim data	ETL, validasi, monitoring	Teknis terbatas

A.6 KPI Utama

KPI	Definisi	Rumus	Sumber Data	Frekuensi Refresh	Pemilik KPI

Contoh KPI:

- total penjualan;
- margin;
- deviasi biaya proyek;
- progres proyek;
- SLA layanan;
- jumlah pelanggan aktif;
- data error rate.

A.7 Risiko Awal

Risiko	Dampak	Kemungkinan	Mitigasi
Kualitas data buruk	Dashboard tidak dipercaya	Tinggi / Sedang / Rendah	Data profiling dan cleansing
Scope melebar	Biaya dan waktu naik		Batasi MVP
Data sensitif bocor	Risiko hukum dan reputasi		RBAC, masking, audit log
ETL gagal	Data tidak terbarui		Logging, retry, monitoring
User tidak memakai dashboard	Nilai bisnis rendah		UAT dan training

A.8 Estimasi Biaya

Komponen	Estimasi Biaya
SDM internal	
Konsultan / pendamping	
Platform / infrastruktur	
Storage	
Backup	
Monitoring	
Security review	
Training	
Dokumentasi	
Cadangan risiko	
Total estimasi	

Rumus sederhana:

Total Biaya = Implementasi Awal + Operasional + Cadangan Risiko

A.9 Tim Proyek

Peran	Nama / Unit	Tanggung Jawab
Sponsor bisnis		Mandat, anggaran, prioritas
Product owner data		Kebutuhan dan prioritas data
Data architect		Arsitektur dan model data
Data engineer		Pipeline ETL/ELT
DBA / platform engineer		Database, backup, tuning
BI developer		Dashboard dan laporan
Data steward		Kualitas dan definisi data
Security reviewer		Akses, masking, compliance
Project manager		Jadwal, risiko, koordinasi

A.10 Timeline

Fase	Aktivitas	Target Waktu	Output
1	Assessment kebutuhan dan sumber data		Requirement brief
2	Desain arsitektur dan model		Arsitektur dan star schema
3	ETL/ELT dan staging		Pipeline awal
4	Data mart dan dashboard		Dashboard MVP
5	Security, governance, testing		Access matrix dan test report
6	UAT dan deployment		Sign-off dan production
7	Dokumentasi dan training		Manual dan pelatihan

A.11 Kriteria Sukses

Kriteria	Target
Data tersedia	Data utama berhasil dimuat ke DW
KPI tervalidasi	Angka sesuai persetujuan data owner
Dashboard digunakan	Digunakan oleh target pengguna
ETL stabil	Berjalan sesuai jadwal
Kualitas data terukur	Error dicatat dan ditangani
Keamanan aktif	Role, masking, dan audit log tersedia
Backup tersedia	Backup berhasil dan restore test dijadwalkan
Dokumentasi lengkap	Data dictionary, lineage, ETL mapping, operational manual tersedia
UAT disetujui	Ada business sign-off
Scope terkendali	MVP selesai sesuai batasan awal

A.12 Persetujuan

Nama	Peran	Tanda Tangan	Tanggal
	Sponsor bisnis		
	Product owner data		
	Project manager		
	Data owner		
	Security / compliance reviewer		

Catatan akhir: *Project Charter* ini menjadi acuan awal proyek. Setiap perubahan besar pada **scope, biaya, KPI, keamanan, atau timeline** harus disetujui dan didokumentasikan.

Lampiran B. Template Requirement dan Data Source Assessment

B.1 Identitas Assessment

Komponen	Isi
Nama proyek	
Domain data	
Tanggal assessment	
Unit / divisi	
Penanggung jawab bisnis	
Penanggung jawab teknis	
Tujuan assessment	

B.2 Ringkasan Kebutuhan Data

Pertanyaan	Jawaban
Keputusan bisnis yang ingin didukung	
Laporan / dashboard yang dibutuhkan	
KPI utama	
Periode data historis yang dibutuhkan	
Frekuensi refresh yang diharapkan	Harian / Mingguan / Bulanan / Lainnya
Pengguna utama	Direksi / Manajer / Analis / Auditor / Lainnya
Data sensitif	Ada / Tidak
Kebutuhan audit	Ada / Tidak

B.3 Tabel *Data Source Assessment*

No.	Nama Sumber Data	Pemilik Data	Format	Volume	Frekuensi Update	Kualitas Data	Risiko	Prioritas Integrasi
1			Database / CSV / API / Log / Spreadsheet / Lainnya		Harian / Mingguan / Bulanan / Real-time	Baik / Sedang / Buruk		Tinggi / Sedang / Rendah
2								
3								
4								
5								

B.4 Panduan Pengisian Kolom

Kolom	Cara Mengisi
Nama sumber data	Nama sistem, file, aplikasi, tabel, API, log, atau sumber lain yang akan diintegrasikan.
Pemilik data	Unit atau orang yang bertanggung jawab atas makna, validitas, dan persetujuan data.
Format	Bentuk data: database, CSV, API, log, spreadsheet, JSON, XML, atau format lain.
Volume	Ukuran data atau jumlah baris, misalnya 2 juta transaksi/bulan atau 500 MB/hari.
Frekuensi update	Seberapa sering data berubah atau perlu diambil: harian, mingguan, bulanan, per jam, atau *near-real-time*.
Kualitas data	Penilaian awal: baik, sedang, buruk. Bisa dilihat dari duplikasi, nilai kosong, format tidak konsisten, dan validitas kode.
Risiko	Risiko teknis, bisnis, keamanan, kualitas, akses, atau kepatuhan.
Prioritas integrasi	Tinggi jika penting untuk KPI utama dan tersedia cepat; rendah jika belum kritikal atau kualitasnya buruk.

B.5 Contoh Pengisian

No.	Nama Sumber Data	Pemilik Data	Format	Volume	Frekuensi Update	Kualitas Data	Risiko	Prioritas Integrasi
1	Transaksi Penjualan	Divisi Penjualan	Database	2 juta baris/bulan	Harian	Sedang	Duplikasi transaksi, kode produk tidak konsisten	Tinggi
2	Master Pelanggan	Divisi Pelanggan	CSV	100 ribu baris	Mingguan	Sedang	Nama pelanggan ganda, data kontak sensitif	Tinggi
3	Master Produk	Divisi Operasi	Spreadsheet	10 ribu baris	Bulanan	Baik	Perubahan kategori belum terdokumentasi	Sedang
4	Data Cabang	Divisi Operasi	CSV	500 baris	Saat ada perubahan	Baik	Kode cabang lama masih dipakai	Sedang
5	Log Aplikasi	Tim Platform	Log	5 GB/hari	Harian	Buruk	Format berubah, volume besar	Rendah

B.6 Penilaian Kualitas Data

Gunakan checklist berikut untuk menilai kualitas awal setiap sumber data.

Dimensi Kualitas	Pertanyaan	Status
Completeness	Apakah kolom wajib terisi lengkap?	
Accuracy	Apakah nilai data sesuai kondisi nyata?	
Consistency	Apakah format dan kode konsisten antar periode?	
Timeliness	Apakah data tersedia tepat waktu?	
Uniqueness	Apakah ada duplikasi data?	
Validity	Apakah nilai sesuai aturan bisnis?	

B.7 Skor Prioritas Integrasi

Gunakan skor sederhana untuk menentukan sumber data mana yang diintegrasikan lebih dulu.

Kriteria	Skor 1	Skor 2	Skor 3
Nilai bisnis	rendah	sedang	tinggi
Kesiapan data	buruk	cukup	baik
Kualitas data	banyak masalah	sebagian bermasalah	relatif baik
Kemudahan akses	sulit	sedang	mudah
Risiko keamanan	tinggi	sedang	rendah

Rumus:

> Skor Prioritas = Nilai Bisnis + Kesiapan Data + Kualitas Data + Kemudahan Akses + Risiko Keamanan

Interpretasi:

Total Skor	Prioritas
12–15	**Tinggi**
8–11	**Sedang**
5–7	**Rendah**

B.8 Catatan Risiko dan Tindak Lanjut

Risiko Utama	Dampak	Tindak Lanjut	Penanggung Jawab
Data duplikat	KPI tidak akurat	Buat aturan deduplikasi	
Kolom wajib kosong	Laporan tidak lengkap	Validasi dan perbaikan sumber	
Format berubah	ETL gagal	Buat kontrak format data	
Data sensitif	Risiko kebocoran	Masking dan pembatasan akses	
Tidak ada pemilik data	Isu tidak terselesaikan	Tetapkan *data owner*	

B.9 Keputusan Assessment

Keputusan	Isi
Sumber data prioritas tinggi	
Sumber data ditunda	
Masalah kualitas utama	
Data sensitif yang perlu perlindungan	
Integrasi fase pertama	
Integrasi fase berikutnya	
Catatan persetujuan	

Catatan akhir: hasil assessment ini menjadi dasar untuk **desain arsitektur, desain ETL/ELT, model data, estimasi kapasitas, keamanan, timeline, dan biaya proyek Data Warehouse**.

Lampiran C. Template Dimensional Model

C.1 Identitas Model

Komponen	Isi
Nama domain data	
Nama model	
Data owner	
Data architect	
Tanggal desain	
Versi	
Status	Draft / Review / Disetujui

C.2 Nama *Fact Table*

Komponen	Isi
Nama *fact table*	
Deskripsi	
Proses bisnis yang dimodelkan	
Sumber data utama	
Frekuensi update	Harian / Mingguan / Bulanan / Lainnya
Retensi data	

Contoh:

Komponen	Isi
Nama *fact table*	`FactSales`
Deskripsi	Menyimpan transaksi penjualan detail
Proses bisnis	Penjualan produk ke pelanggan
Sumber data	transaksi penjualan, pelanggan, produk, cabang
Frekuensi update	Harian
Retensi data	5 tahun

C.3 *Grain*

Grain menjelaskan **arti satu baris** pada *fact table*. Ini adalah keputusan paling penting dalam *dimensional modeling*.

Pertanyaan	Jawaban
Satu baris mewakili apa?	
Periode waktu terkecil	
Unit bisnis terkecil	
Apakah data berasal dari satu atau banyak sumber?	
Risiko jika *grain* salah	

Contoh:

Satu baris FactSales = satu item transaksi penjualan pada satu tanggal, untuk satu produk, satu pelanggan, dan satu cabang.

Catatan penting: jangan membuat *fact table* sebelum *grain* disepakati.

C.4 *Measure*

Measure adalah nilai numerik yang dianalisis.

Nama *Measure*	Definisi	Rumus	Tipe Data	Additive?	Catatan
				Ya / Tidak / Semi	

Contoh:

Nama *Measure*	Definisi	Rumus	Tipe Data	Additive ?	Catatan
`quantity`	jumlah unit terjual	dari sumber	numeric	Ya	dapat dijumlahkan
`gross_amount`	nilai sebelum diskon	dari sumber	numeric	Ya	
`discount_amount`	nilai diskon	dari sumber	numeric	Ya	
`net_amount`	nilai setelah diskon	gross - discount	numeric	Ya	KPI utama
`margin_amount`	margin transaksi	net - cost	numeric	Ya	perlu validasi biaya

C.5 *Dimension*

Dimension memberi konteks analisis: siapa, apa, kapan, di mana, melalui kanal apa, atau pada proyek apa.

Nama *Dimension*	Deskripsi	*Key* Utama	Atribut Penting	Sumber Data	SCD Type
					Type 0 / 1 / 2 / 3

Contoh:

Nama *Dimension*	Deskripsi	*Key* Utama	Atribut Penting	Sumber Data	SCD Type
`DimCustomer`	data pelanggan	`customer_key`	nama, tipe, kota, provinsi	master pelanggan	Type 2
`DimProduct`	data produk	`product_key`	nama, kategori, unit	master produk	Type 1
`DimDate`	kalender analitik	`date_key`	tanggal, bulan, kuartal, tahun	generated	Type 0
`DimBranch`	data cabang	`branch_key`	cabang, kota, wilayah	master cabang	Type 1

C.6 *Key*

Gunakan *surrogate key* untuk dimensi agar model stabil dan tidak bergantung penuh pada kode dari sistem sumber.

Tabel	*Surrogate Key*	*Natural Key*	*Foreign Key* di Fact	Catatan

Contoh:

Tabel	*Surrogate Key*	*Natural Key*	*Foreign Key* di Fact	Catatan
`DimCustomer`	`customer_key`	`customer_id`	`customer_key`	mendukung histori pelanggan
`DimProduct`	`product_key`	`product_id`	`product_key`	kode produk dari sumber
`DimDate`	`date_key`	`full_date`	`date_key`	format umum `YYYYMMDD`
`DimBranch`	`branch_key`	`branch_code`	`branch_key`	kode cabang distandardisasi

C.7 SCD Type

SCD atau *Slowly Changing Dimension* digunakan untuk mengelola perubahan atribut dimensi.

Dimension	**Atribut yang Berubah**	**SCD Type**	**Alasan**	**Catatan Implementasi**
		Type 0 / 1 / 2 / 3		

Panduan singkat:

SCD Type	**Fungsi**	**Cocok Untuk**
Type 0	tidak berubah	tanggal lahir, kode tetap
Type 1	nilai lama ditimpa	koreksi nama, koreksi ejaan
Type 2	histori disimpan sebagai baris baru	alamat, segmen pelanggan, wilayah
Type 3	nilai lama disimpan di kolom tambahan	perubahan terbatas yang hanya butuh histori pendek

C.8 *Business Rule*

Business rule menjelaskan aturan perhitungan, validasi, dan interpretasi data.

No.	*Business Rule*	Berlaku Untuk	Dampak ke Model / ETL
1			
2			
3			

Contoh:

No.	*Business Rule*	Berlaku Untuk	Dampak ke Model / ETL
1	`net_amount = gross_amount - discount_amount`	`FactSales`	dihitung saat load fakta
2	transaksi dengan `quantity <= 0` masuk tabel error	`FactSales`	validasi sebelum load
3	pelanggan tanpa `customer_id` ditolak	`DimCustomer`	wajib ada *natural key*
4	produk tidak dikenal masuk daftar pengecualian	`DimProduct`	perlu proses kualitas data
5	margin negatif diberi flag untuk review	`FactSales`	tidak selalu ditolak, tetapi diaudit

C.9 Ringkasan Persetujuan Model

Komponen	Keputusan
Fact table utama	
Grain disetujui	Ya / Tidak
Measure utama	
Dimension utama	
Key utama	
SCD digunakan	
Business rule kritis	
Risiko desain	
Status persetujuan bisnis	Disetujui / Revisi / Ditunda

C.10 Persetujuan

Nama	Peran	Tanda Tangan	Tanggal
	Data owner		
	Data architect		
	Data steward		
	Product owner data		
	Security reviewer		

Catatan akhir: kesalahan pada **grain, measure, dimension, key, SCD, atau business rule** akan berdampak langsung pada akurasi dashboard, kualitas audit, dan kepercayaan pengguna.

Lampiran D. Rumus Perencanaan Kapasitas

D.1 Rumus Ukuran Data

Rumus dasar:

> Ukuran Data = Jumlah Baris × Ukuran Rata-Rata per Baris

Contoh:

> Jumlah baris = 2.000.000 transaksi/bulan
> Ukuran rata-rata = 500 byte
>
> Ukuran Data = 2.000.000 × 500 byte
> = 1.000.000.000 byte
> ≈ 1 GB/bulan

Catatan:

- Gunakan ukuran rata-rata baris dari hasil *data profiling*.
- Jangan hanya menghitung *fact table*; perhitungkan juga dimensi, indeks, staging, log, dan backup.

D.2 Rumus *Growth* Data

Rumus pertumbuhan tahunan:

> Data Tahun ke-n = Data Awal × (1 + Growth Rate)^n

Contoh:

> Data awal = 10 TB
> Growth = 30% per tahun
> Periode = 3 tahun
>
> Data Tahun ke-3 = 10 × (1 + 0,30)^3
> = 10 × 2,197
> = 21,97 TB

Untuk total akumulasi beberapa tahun:

> Total Data Retensi = Data Tahun 1 + Data Tahun 2 + ... + Data Tahun n

D.3 Rumus *Storage Total*

Rumus umum:

Storage Total =
Raw Data
+ Staging
+ Warehouse
+ Data Mart
+ Index
+ Materialized View
+ Log
+ Backup
+ Archive
+ Cadangan

Rumus praktis:

Storage Total = Subtotal Storage × (1 + Cadangan Kapasitas)

Contoh cadangan:

Cadangan Kapasitas = 20% sampai 30%

D.4 Rumus Backup

Rumus dasar:

Backup = Data yang Dibackup × Jumlah Salinan Backup

Rumus lebih lengkap:

Backup Total =
(Warehouse + Index + Metadata + ETL Script + Dashboard Config + Log Penting)
× Jumlah Salinan

Contoh:

Warehouse + Index = 20 TB
Backup = 2 salinan

Backup Total = 20 × 2
= 40 TB

Catatan penting: backup harus mencakup **data, metadata, konfigurasi, script ETL, dan dokumen operasional**.

D.5 Rumus *Retention*

Retention menentukan berapa lama data disimpan aktif dan berapa lama data diarsipkan.

Data Retensi = Data Tahunan × Jumlah Tahun Retensi

Jika ada growth:

Data Retensi = Σ Data Tahun ke-n selama periode retensi

Contoh:

Tahun	Data
Tahun 1	10 TB
Tahun 2	13 TB
Tahun 3	16,9 TB
Tahun 4	21,97 TB
Tahun 5	28,56 TB

Total Retensi 5 Tahun = 10 + 13 + 16,9 + 21,97 + 28,56
= 90,43 TB

D.6 Rumus Compute

Rumus sederhana:

Kebutuhan Compute =
Beban Query
+ Beban ETL
+ Beban Dashboard
+ Beban Data Quality
+ Beban Maintenance

Rumus skor awal:

Beban Query = Jumlah User Aktif × Query per User × Skor Kompleksitas Query

Skor kompleksitas:

Jenis Query	Skor
Dashboard ringan	1
Laporan standar	2
Query banyak join	3
Agregasi besar	4
Query eksploratif berat	5

Contoh:

User aktif = 20
Query per user = 3
Skor kompleksitas = 2

Beban Query = 20 × 3 × 2
= 120

D.7 Rumus *Concurrency*

Concurrency menghitung beban pengguna aktif bersamaan.

Concurrency Rate = User Aktif Bersamaan / Total User Terdaftar

Contoh:

Total user = 200
User aktif bersamaan = 25

Concurrency Rate = 25 / 200
= 12,5%

Rumus beban bersamaan:

Beban Concurrency =
User Aktif Bersamaan × Rata-Rata Query per User × Kompleksitas Query

Interpretasi awal:

Beban	Interpretasi
0–50	Ringan
51–150	Sedang
151–300	Berat
>300	Perlu pemisahan workload / scaling

D.8 Rumus *Network Transfer*

Rumus dasar:

> Network Transfer = Volume Data × Frekuensi Transfer

Jika data dikirim dari beberapa sumber:

> Total Transfer =
> Transfer Sumber 1 + Transfer Sumber 2 + ... + Transfer Sumber n

Contoh:

> Data harian = 100 GB
> Frekuensi = 30 hari
>
> Transfer Bulanan = 100 GB × 30
> = 3.000 GB
> = 3 TB/bulan

Jika ada backup ke lokasi cadangan:

> Transfer Backup = Ukuran Backup × Frekuensi Backup

Catatan penting: gunakan *incremental load* dan kompresi untuk mengurangi transfer yang tidak perlu.

D.9 Rumus *Query Cost*

Rumus biaya per query:

Biaya per Query = Total Biaya Bulanan / Jumlah Query Bulanan

Contoh:

Total biaya bulanan = Rp120.000.000
Jumlah query bulanan = 1.200.000

Biaya per Query = Rp120.000.000 / 1.200.000
= Rp100/query

Rumus biaya query per kategori:

Biaya Query Kategori =
Biaya Workload Kategori / Jumlah Query Kategori

Contoh kategori:

- dashboard;
- laporan standar;
- query analis;
- audit;
- ETL validation.

D.10 Rumus *Energy per Query*

Rumus dasar:

Energi per Query = Total Energi Bulanan / Jumlah Query Bulanan

Contoh:

Energi bulanan = 2.304 kWh
Jumlah query bulanan = 1.000.000

Energi per Query = 2.304 / 1.000.000
= 0,002304 kWh/query

Jika ingin memasukkan efisiensi fasilitas:

Energi Fasilitas = Energi IT × PUE

Kemudian:

Energi per Query = Energi Fasilitas / Jumlah Query

D.11 Template Ringkas Perhitungan Kapasitas

Komponen	Rumus	Hasil
Ukuran data	jumlah baris × ukuran baris	
Growth data	data awal × (1 + growth)^n	
Retensi	total data selama periode retensi	
Storage total	raw + staging + warehouse + index + backup + archive	
Backup	data dibackup × jumlah salinan	
Compute	user aktif × query/user × kompleksitas	
Concurrency	user aktif bersamaan / total user	
Network transfer	volume data × frekuensi transfer	
Query cost	biaya bulanan / jumlah query	
Energy per query	energi bulanan / jumlah query	

D.12 Catatan Praktis

- **Hitung kapasitas berdasarkan pertumbuhan 3–5 tahun**, bukan hanya kondisi awal.
- **Backup, indeks, staging, log, dan archive wajib masuk perhitungan.**
- **Compute dihitung dari user aktif bersamaan**, bukan hanya total user.
- **Query cost** membantu melihat efisiensi pemakaian sistem.
- **Energy per query** membantu menghubungkan optimasi data dengan efisiensi energi.
- Tambahkan **cadangan kapasitas 20–30%** untuk pertumbuhan tak terduga, perubahan scope, dan kebutuhan audit.

Lampiran E. Template Perhitungan Biaya

E.1 Identitas Perhitungan

Komponen	Isi
Nama proyek	
Domain data	
Periode estimasi	Bulanan / Tahunan / 3 Tahun
Tanggal perhitungan	
Penyusun	
Versi	
Catatan asumsi utama	

E.2 Template Perhitungan Biaya

No.	Komponen Biaya	Satuan	Volume	Harga Satuan	Biaya Bulanan	Biaya Tahunan	Catatan
1	Storage data aktif	TB/bulan					Warehouse, data mart
2	Storage raw / archive	TB/bulan					Data mentah dan historis
3	Storage staging	TB/bulan					Area sementara ETL/ELT
4	Storage backup	TB/bulan					Termasuk jumlah salinan
5	Compute ETL/ELT	jam/bulan					Extract, transform, load
6	Compute dashboard	jam/bulan					Query dashboard dan laporan
7	Compute analis	jam/bulan					Query eksploratif
8	Network transfer	GB/bulan					Transfer data sumber, backup, replikasi
9	Monitoring	bulan					Log, alert, dashboard teknis
10	Security operation	bulan					Audit log, masking, review akses
11	Backup operation	bulan					Jadwal backup dan restore test
12	*Disaster recovery*	bulan					Secondary site, drill, restore
13	Dukungan *open source*	bulan					Support, tuning, patching
14	SDM internal	orang-bulan					Tim proyek / operasi
15	Konsultan	orang-bulan					Arsitektur, ETL, tuning
16	Training	paket / peserta					Tim teknis dan pengguna
17	Dokumentasi	paket					Manual, lineage, data dictionary
18	UAT dan workshop	sesi					Validasi bisnis
19	Maintenance	bulan					Vacuum, index, partition, cleanup
20	Cadangan risiko	%					10–20% dari subtotal
	TOTAL						

E.3 Rumus Perhitungan

Biaya Bulanan = Volume × Harga Satuan

Biaya Tahunan = Biaya Bulanan × 12

Subtotal = Jumlah seluruh komponen biaya

Cadangan Risiko = Subtotal × Persentase Cadangan Risiko

Total Estimasi = Subtotal + Cadangan Risiko

Untuk estimasi 3 tahun:

TCO 3 Tahun = Biaya Implementasi Awal + (Biaya Operasional Bulanan × 36) + Cadangan Risiko

E.4 Contoh Pengisian Ringkas

No.	Komponen Biaya	Satuan	Volume	Harga Satuan	Biaya Bulanan	Biaya Tahunan	Catatan
1	Storage data aktif	TB/bulan	10	Rp500.000	Rp5.000.000	Rp60.000.000	Warehouse dan data mart
2	Storage backup	TB/bulan	20	Rp300.000	Rp6.000.000	Rp72.000.000	2 salinan backup
3	Compute ETL/ELT	jam/bulan	120	Rp100.000	Rp12.000.000	Rp144.000.000	Refresh harian
4	Compute dashboard	jam/bulan	200	Rp100.000	Rp20.000.000	Rp240.000.000	50 user BI
5	Monitoring	bulan	1	Rp5.000.000	Rp5.000.000	Rp60.000.000	Alert dan log
6	Security operation	bulan	1	Rp8.000.000	Rp8.000.000	Rp96.000.000	Review akses dan audit
7	SDM operasi	orang-bulan	2	Rp25.000.000	Rp50.000.000	Rp600.000.000	Data engineer dan DBA
	Subtotal				**Rp106.000.000**	**Rp1.272.000.000**	
	Cadangan risiko 15%	%	15%		**Rp15.900.000**	**Rp190.800.000**	
	Total				**Rp121.900.000**	**Rp1.462.800.000**	

E.5 Catatan Asumsi

Asumsi	Nilai
Data awal	
Growth data per tahun	
Retensi data	
Jumlah user BI	
User aktif bersamaan	
Frekuensi refresh	
Jumlah salinan backup	
Cadangan risiko	
Target SLA	

E.6 Checklist Validasi Biaya

Pertanyaan	Ya/Tidak	Catatan
Apakah storage aktif sudah dihitung?		
Apakah raw, staging, dan archive masuk perhitungan?		
Apakah backup dan jumlah salinannya dihitung?		
Apakah compute ETL dan dashboard dipisahkan?		
Apakah biaya monitoring dihitung?		
Apakah biaya security dihitung?		
Apakah biaya SDM internal dihitung?		
Apakah training dan UAT dihitung?		
Apakah maintenance dan operasi bulanan dihitung?		
Apakah cadangan risiko ditambahkan?		

Catatan akhir: estimasi biaya *Data Warehouse* harus direview berkala karena volume data, jumlah pengguna, frekuensi refresh, kebutuhan backup, dan kompleksitas query dapat berubah setelah sistem masuk *production*.

Lampiran F. Template Perencanaan SDM

F.1 Identitas Perencanaan SDM

Komponen	Isi
Nama proyek	
Domain data	
Fase proyek	Assessment / Desain / Implementasi / Deployment / Operasi
Durasi proyek	
Tanggal perencanaan	
Penyusun	
Versi	

F.2 Template Perencanaan SDM

No.	Peran	Jumlah Orang	Kompetensi	Durasi	Biaya per Bulan	Total Biaya	Risiko Jika Tidak Tersedia
1	Sponsor bisnis		Mandat, prioritas bisnis, pengambilan keputusan				Proyek tidak punya arah dan dukungan organisasi
2	*Product owner data*		Requirement, prioritas KPI, komunikasi bisnis-teknis				Kebutuhan bisnis tidak jelas dan scope mudah melebar
3	*Data architect*		Arsitektur DW, *dimensional modeling*, metadata				Model data salah, sulit dikembangkan, dan tidak konsisten
4	*Data engineer*		ETL/ELT, *pipeline*, staging, rekonsiliasi				Data tidak mengalir stabil dan proses refresh gagal
5	*ETL developer*		Extract, transform, load, error handling, logging				Integrasi data lambat, banyak error, sulit diaudit
6	DBA / *platform engineer*		Database, backup, tuning, monitoring, restore				Sistem lambat, backup lemah, risiko gagal produksi
7	*BI developer*		Dashboard, visualisasi, query KPI, data mart				Data tidak mudah dipakai pengguna bisnis
8	*Data analyst*		Analisis data, validasi KPI, UAT, insight bisnis				Dashboard kurang relevan dan angka tidak tervalidasi
9	*Data steward*		Kualitas data, definisi KPI, metadata, data dictionary				Definisi data kacau, kualitas rendah, konflik antar unit
10	*Security / compliance reviewer*		RBAC, masking, audit log, klasifikasi data, compliance				Data sensitif berisiko bocor dan audit gagal
11	*Project manager*		Jadwal, risiko, koordinasi, dokumentasi, komunikasi				Proyek terlambat, tidak terkendali, dan dokumentasi lemah
12	Trainer / pendamping pengguna		Pelatihan dashboard, adopsi, panduan penggunaan				Sistem selesai tetapi tidak digunakan optimal

F.3 Rumus Biaya SDM

Total Biaya per Peran = Jumlah Orang × Durasi Bulan × Biaya per Bulan

Total Biaya SDM = Jumlah seluruh total biaya per peran

Total Realistis = Total Biaya SDM + Cadangan Risiko

Rekomendasi cadangan risiko:

Cadangan Risiko = 10%–20% × Total Biaya SDM

F.4 Contoh Pengisian Ringkas

Peran	Jumlah Orang	Kompetensi	Durasi	Biaya per Bulan	Total Biaya	Risiko Jika Tidak Tersedia
Data architect	1	Arsitektur, model dimensional, metadata	3 bulan	Rp35 juta	Rp105 juta	Desain tidak konsisten
Data engineer	2	ETL/ELT, staging, pipeline	4 bulan	Rp25 juta	Rp200 juta	Data refresh tidak stabil
DBA / *platform engineer*	1	Database, backup, tuning	2 bulan	Rp25 juta	Rp50 juta	Sistem lambat dan sulit dipulihkan
BI developer	1	Dashboard, data mart, query KPI	3 bulan	Rp20 juta	Rp60 juta	Dashboard tidak siap digunakan
Data steward	1	Kualitas data, definisi KPI	3 bulan	Rp18 juta	Rp54 juta	Data tidak dipercaya
Project manager	1	Koordinasi, risiko, jadwal	4 bulan	Rp25 juta	Rp100 juta	Proyek tidak terkendali
Total					**Rp569 juta**	

F.5 Matriks Peran yang Bisa Dirangkap

Peran	Bisa Dirangkap?	Catatan
Sponsor bisnis	Tidak ideal	Harus punya kewenangan keputusan
Product owner data	Ya	Bisa dirangkap oleh wakil bisnis yang kuat
Data architect	Ya	Bisa merangkap reviewer desain, bukan semua teknis
Data engineer	Tidak disarankan	Beban tinggi dan kritikal
DBA / *platform engineer*	Ya	Bisa dirangkap jika sistem kecil
BI developer	Ya	Bisa dirangkap analis pada proyek kecil
Data steward	Ya	Bisa paruh waktu dari unit bisnis
Security reviewer	Ya	Tetapi tetap harus independen saat review
Project manager	Ya	Untuk proyek kecil, bisa dirangkap product owner

F.6 Checklist Kesiapan SDM

Pertanyaan	Ya/Tidak	Catatan
Apakah sponsor bisnis sudah jelas?		
Apakah pemilik KPI sudah ditetapkan?		
Apakah *data architect* tersedia?		
Apakah *data engineer* cukup untuk jumlah sumber data?		
Apakah ada pengelola database dan backup?		
Apakah ada pembuat dashboard?		
Apakah ada *data steward* untuk kualitas data?		
Apakah ada reviewer keamanan dan compliance?		
Apakah ada PM atau koordinator proyek?		
Apakah biaya SDM sudah dihitung lengkap?		
Apakah risiko kekurangan SDM sudah dimitigasi?		

F.7 Catatan Risiko SDM

Risiko	Dampak	Mitigasi
Data engineer kurang	ETL lambat dan tidak stabil	Kurangi scope, tambah SDM, gunakan fase bertahap
Tidak ada *data steward*	KPI tidak konsisten	Tunjuk steward dari unit bisnis
Tidak ada DBA	Backup, tuning, dan restore lemah	Libatkan platform engineer
Tidak ada security reviewer	Risiko kebocoran data	Wajibkan review akses dan masking
Tidak ada PM	Jadwal dan risiko tidak terkendali	Tetapkan koordinator proyek
User bisnis tidak terlibat	Dashboard tidak dipakai	Jadwalkan UAT dan workshop

Catatan akhir: perencanaan SDM harus disesuaikan dengan **skala proyek, jumlah sumber data, risiko data sensitif, target deployment, dan kebutuhan operasi setelah production**.

Lampiran G. Checklist Deployment dan Operasi

G.1 Identitas Deployment

Komponen	Isi
Nama proyek	
Domain data	
Environment target	*Production*
Tanggal deployment	
Versi release	
Penanggung jawab teknis	
Penanggung jawab bisnis	
Strategi cutover	*Big bang* / *Parallel run* / *Phased rollout* / *Pilot unit*

G.2 Checklist Deployment

No.	Checklist	Status	Catatan / Bukti
1	**Schema siap**	Ya / Tidak	Skema `raw`, `staging`, `dw`, `mart`, dan `audit` sudah dibuat
2	**ETL sukses**	Ya / Tidak	Semua proses *extract*, *transform*, dan *load* selesai tanpa error kritikal
3	**Data valid**	Ya / Tidak	Row count, null check, duplicate check, dan rekonsiliasi sudah lolos
4	**Dashboard valid**	Ya / Tidak	KPI, filter, grafik, dan angka dashboard sudah di cek pengguna bisnis
5	**User access benar**	Ya / Tidak	Role, hak akses, masking, dan pembatasan ekspor sudah diuji
6	**Backup aktif**	Ya / Tidak	Backup awal tersedia dan jadwal backup sudah berjalan
7	**Monitoring aktif**	Ya / Tidak	Monitoring ETL, query, storage, backup, dan error sudah tersedia
8	**Dokumentasi selesai**	Ya / Tidak	*Data dictionary*, *ETL mapping*, *lineage*, dan manual operasi tersedia
9	**UAT disetujui**	Ya / Tidak	Ada *business sign-off* dari pengguna dan *data owner*
10	**Rollback plan tersedia**	Ya / Tidak	Snapshot, backup, prosedur restore, dan kontak darurat tersedia

G.3 Detail Validasi Minimum

Area	Pemeriksaan Minimum	Bukti
Schema	tabel dimensi, fakta, staging, audit, dan log tersedia	DDL / daftar tabel
ETL	job berjalan sesuai urutan dan jadwal	ETL log
Data	jumlah baris sumber dan target sesuai	hasil rekonsiliasi
Dashboard	angka KPI sesuai definisi bisnis	hasil UAT
Access	user hanya melihat data sesuai role	access matrix
Backup	backup berhasil dibuat	backup log
Monitoring	alert dan dashboard teknis aktif	monitoring screenshot / log
Dokumentasi	dokumen utama tersedia dan terbaru	daftar dokumen
UAT	pengguna menyetujui hasil	form sign-off
Rollback	prosedur pemulihan tertulis	rollback document

G.4 Checklist Operasi Hari Pertama

Pemeriksaan	Status	Catatan
ETL pertama setelah deployment berjalan sukses		
Data refresh sesuai jadwal		
Dashboard dapat diakses pengguna		
Tidak ada error kritikal di log		
Query dashboard dalam batas SLA		
Storage masih aman		
Backup setelah deployment sukses		
Monitoring mengirim alert dengan benar		
Tidak ada akses mencurigakan		
Isu pengguna dicatat dan ditindaklanjuti		

G.5 Kriteria *Go / No-Go*

Kriteria	Keputusan
Semua checklist kritikal berstatus **Ya**	Go
Ada error data yang memengaruhi KPI utama	No-Go
Backup belum tersedia	No-Go
User access belum diuji	No-Go
UAT belum disetujui	No-Go
Rollback plan tidak tersedia	No-Go
Dokumentasi operasi belum tersedia	Go bersyarat / No-Go sesuai risiko

G.6 Persetujuan Deployment

Nama	Peran	Keputusan	Tanda Tangan	Tanggal
	Sponsor bisnis	Go / No-Go		
	Data owner	Go / No-Go		
	Data architect	Go / No-Go		
	Data engineer	Go / No-Go		
	Security reviewer	Go / No-Go		
	Project manager	Go / No-Go		

Catatan akhir: *Deployment* hanya boleh dilakukan jika **schema siap, ETL sukses, data valid, dashboard valid, user access benar, backup aktif, monitoring aktif, dokumentasi selesai, UAT disetujui, dan rollback plan tersedia**.

Lampiran H. 30 Soal Simulasi Sertifikasi *Data Warehouse*

H.1 Soal Pilihan Ganda

Pilih satu jawaban yang paling tepat.

No.	Soal	A	B	C	D
1	Tujuan utama *Data Warehouse* adalah…	transaksi harian	analitik dan keputusan	input data	pengganti aplikasi operasional
2	OLTP paling cocok untuk…	laporan historis	transaksi operasional	agregasi tahunan	analitik multidimensi
3	OLAP paling cocok untuk…	input transaksi	update stok real-time	analisis tren dan KPI	validasi login
4	Ciri DW yang menyimpan histori disebut…	volatile	time-variant	temporary	transactional
5	Area sementara sebelum data masuk DW disebut…	dashboard	staging	archive	report
6	Proses mengambil, mengubah, lalu memuat data disebut…	OLAP	ETL	KPI	RBAC
7	Dalam model dimensional, tabel berisi transaksi numerik disebut…	dimension table	fact table	metadata table	access table
8	*Grain* menjelaskan…	tipe user	ukuran storage	arti satu baris fact table	jumlah dashboard

9	`DimCustomer`, `DimProduct`, dan `DimDate` adalah contoh…	measure	dimension	backup	query
10	Skema dengan fact table di tengah dan dimension di sekeliling disebut…	star schema	network schema	raw schema	audit schema
11	SCD Type 2 digunakan untuk…	menghapus histori	menyimpan histori perubahan	menolak data	mempercepat backup
12	Metadata menjelaskan…	data tentang data	data rusak	data sementara	data tidak terpakai
13	*Data lineage* berguna untuk mengetahui…	harga server	asal dan alur data	jumlah user	biaya listrik
14	Prinsip *least privilege* berarti…	semua user bebas akses	akses minimum sesuai tugas	akses admin untuk semua	akses tanpa audit
15	RBAC adalah kontrol akses berdasarkan…	ukuran data	role pengguna	nama tabel	tipe storage
16	*Data masking* digunakan untuk…	memperbesar data	menyamarkan data sensitif	menghapus backup	membuat indeks
17	*Partitioning* paling berguna untuk…	tabel kecil	tabel besar historis	dokumen manual	user training
18	*Materialized view* digunakan untuk…	menyimpan hasil query agar cepat	menghapus data	mengganti backup	membuat user
19	RPO menjelaskan…	target kehilangan data maksimum	target biaya bulanan	jumlah query	ukuran tabel
20	TCO adalah…	total biaya kepemilikan	total query output	tipe constraint	tabel data operasional

H.2 Soal Studi Kasus Singkat

Jawab ringkas, fokus pada alasan desain.

No.	Studi Kasus	Pertanyaan
21	Sebuah organisasi memiliki data penjualan, pelanggan, produk, dan cabang. Direksi ingin dashboard penjualan bulanan.	Buat rancangan *star schema* minimal.
22	Dashboard sering berbeda angkanya dengan laporan lama.	Sebutkan 5 langkah validasi data yang harus dilakukan.
23	Analis meminta akses ke semua data pelanggan termasuk nomor identitas.	Apa kebijakan akses yang aman?
24	ETL harian sering gagal karena kode produk baru belum ada di master data.	Apa perbaikan proses yang disarankan?
25	Organisasi ingin go-live DW dalam 90 hari.	Susun fase implementasi utama dari assessment sampai review.

H.3 Soal Perhitungan Kapasitas / Biaya

Kerjakan dengan rumus sederhana dan tuliskan asumsi.

No.	Soal	Rumus yang Dipakai
26	Terdapat 3 juta transaksi per bulan. Ukuran rata-rata 1 baris = 400 byte. Hitung ukuran data per bulan.	`Ukuran Data = jumlah baris × ukuran baris`
27	Data awal 10 TB, growth 25% per tahun. Hitung estimasi data tahun ke-3.	`Data Tahun ke-n = data awal × (1 + growth)^n`
28	Warehouse aktif 20 TB, index overhead 30%, backup 2 salinan. Hitung kebutuhan backup untuk warehouse + index.	`Backup = (warehouse + index) × jumlah salinan`
29	Total biaya bulanan DW Rp150 juta. Jumlah query bulanan 1,5 juta. Hitung biaya per query.	`Biaya per Query = biaya bulanan / jumlah query`
30	Biaya implementasi awal Rp900 juta. OPEX bulanan Rp125 juta. Hitung TCO 3 tahun sebelum cadangan risiko.	`TCO 3 Tahun = implementasi awal + (OPEX × 36)`

H.4 Kunci Jawaban Pilihan Ganda

No.	Jawaban	No.	Jawaban	No.	Jawaban	No.	Jawaban
1	B	6	B	11	B	16	B
2	B	7	B	12	A	17	B
3	C	8	C	13	B	18	A
4	B	9	B	14	B	19	A
5	B	10	A	15	B	20	A

H.5 Kunci Ringkas Perhitungan

No.	Jawaban Ringkas
26	`3.000.000 × 400 = 1.200.000.000 byte ≈ 1,2 GB/bulan`
27	`10 × 1,25³ = 19,53 TB`
28	`Index = 20 × 30% = 6 TB`; backup = `(20 + 6) × 2 = 52 TB`
29	`Rp150.000.000 / 1.500.000 = Rp100/query`
30	`Rp900.000.000 + (Rp125.000.000 × 36) = Rp5.400.000.000`

Catatan: soal studi kasus dinilai dari **ketepatan konsep, alasan desain, kelengkapan kontrol, dan kemampuan menghubungkan kebutuhan bisnis dengan arsitektur DW**.

Glossary

Istilah	Definisi Singkat
Access Matrix	Tabel yang menjelaskan siapa boleh mengakses data apa, dengan hak baca, tulis, admin, ekspor, dan akses data sensitif.
Aggregation	Proses meringkas data detail menjadi data ringkasan, misalnya penjualan harian menjadi penjualan bulanan.
Archive	Penyimpanan data lama yang jarang diakses tetapi masih perlu disimpan untuk audit, historis, atau kepatuhan.
Audit Log	Catatan aktivitas sistem, seperti siapa mengakses data, kapan, dari mana, dan tindakan apa yang dilakukan.
Audit Readiness	Kesiapan sistem untuk diperiksa auditor, termasuk bukti akses, perubahan data, asal data, dan perlindungan data sensitif.
Backup	Salinan data, metadata, konfigurasi, dan script penting untuk pemulihan jika terjadi kerusakan atau kehilangan data.
BI Developer	Peran yang bertanggung jawab membuat laporan, *dashboard*, visualisasi, dan tampilan analitik untuk pengguna bisnis.
Business Sign-off	Persetujuan resmi dari pemilik bisnis bahwa data, laporan, dan dashboard sudah benar dan siap digunakan.
CAPEX	*Capital expenditure*, yaitu belanja modal di awal seperti server, storage, perangkat jaringan, dan fasilitas.
CDC	*Change Data Capture*, teknik menangkap perubahan data dari sistem sumber, seperti insert, update, dan delete.
Cloud Data Warehouse	*Data Warehouse* yang berjalan pada layanan *cloud*, biasanya dengan pemisahan antara storage dan compute.
Compute	Sumber daya pemrosesan yang digunakan untuk menjalankan query, ETL, transformasi, agregasi, dan dashboard.
Concurrency	Kemampuan sistem melayani banyak query atau pengguna secara bersamaan.
Confidential Data	Data rahasia organisasi yang jika bocor dapat merugikan bisnis, seperti transaksi detail, margin, kontrak, dan biaya.
Cost Planning	Perencanaan biaya implementasi dan operasi *Data Warehouse*, termasuk SDM, storage, compute, backup, security, dan monitoring.

Cutover	Proses perpindahan dari sistem lama atau proses lama ke sistem *Data Warehouse* baru.
Dashboard	Tampilan visual yang menyajikan KPI, tren, grafik, tabel, dan ringkasan informasi untuk pengambilan keputusan.
Data Analyst	Peran yang menganalisis data, membuat insight, memvalidasi KPI, dan membantu pengguna memahami hasil analitik.
Data Architect	Peran yang merancang arsitektur, model data, skema, integrasi, metadata, dan standar desain *Data Warehouse*.
Data Classification	Pengelompokan data berdasarkan tingkat sensitivitas, misalnya Public, Internal, Confidential, Restricted, dan Personal Data.
Data Cleansing	Proses membersihkan data dari kesalahan, duplikasi, format tidak konsisten, nilai kosong, atau data tidak valid.
Data Dictionary	Dokumen yang menjelaskan arti tabel, kolom, tipe data, aturan bisnis, pemilik data, dan catatan penting lainnya.
Data Engineer	Peran yang membangun pipeline, ETL/ELT, staging, transformasi, validasi, dan proses load ke *Data Warehouse*.
Data Governance	Tata kelola data yang mengatur kepemilikan, standar, kualitas, keamanan, akses, metadata, dan kepatuhan data.
Data Lineage	Jejak asal-usul dan perjalanan data dari sumber, staging, transformasi, warehouse, data mart, sampai dashboard.
Data Mart	Subset *Data Warehouse* yang disiapkan untuk kebutuhan unit, domain, atau laporan tertentu.
Data Masking	Teknik menyamarkan data sensitif agar pengguna tetap bisa bekerja tanpa melihat nilai asli.
Data Mining	Proses menemukan pola, hubungan, tren, atau informasi tersembunyi dari data dalam jumlah besar.
Data Owner	Pihak bisnis yang bertanggung jawab atas makna, kualitas, penggunaan, dan persetujuan data pada domain tertentu.
Data Profiling	Proses memeriksa kondisi data sumber, seperti jumlah baris, nilai kosong, duplikasi, format, dan anomali.
Data Quality	Tingkat kelayakan data berdasarkan completeness, accuracy, consistency, timeliness, uniqueness, dan validity.
Data Steward	Peran yang menjaga definisi data, kualitas data, metadata, aturan bisnis, dan koordinasi isu data harian.

Data Warehouse	Sistem terintegrasi untuk menyimpan data historis dari berbagai sumber guna mendukung analitik, laporan, audit, dan keputusan.
DBA	*Database Administrator*, peran yang mengelola database, performa, backup, restore, keamanan teknis, dan maintenance.
Deployment	Proses membawa sistem dari development atau staging ke production agar siap dipakai pengguna resmi.
Development Environment	Lingkungan kerja untuk membangun, mencoba, dan mengembangkan tabel, ETL, query, dan dashboard.
Dimension Table	Tabel yang berisi konteks analisis, seperti pelanggan, produk, cabang, waktu, vendor, atau proyek.
Dimensional Modeling	Teknik pemodelan data untuk analitik menggunakan fact table dan dimension table.
Disaster Recovery	Rencana pemulihan sistem setelah gangguan besar, seperti kerusakan server, kehilangan data, atau bencana.
Drill-down	Operasi analitik untuk melihat data dari ringkasan menuju detail, misalnya dari tahun ke bulan.
ELT	*Extract, Load, Transform*, proses memuat data ke platform target terlebih dahulu, lalu melakukan transformasi di dalamnya.
Encryption at Rest	Enkripsi data saat disimpan, misalnya di database, storage, backup, atau archive.
Encryption in Transit	Enkripsi data saat dikirim melalui jaringan antara sistem sumber, warehouse, dashboard, atau API.
ETL	*Extract, Transform, Load*, proses mengambil data dari sumber, mengubahnya, lalu memuatnya ke *Data Warehouse*.
ETL Mapping	Dokumen yang menjelaskan pemetaan kolom dari sumber ke target serta aturan transformasinya.
Executive Dashboard	Dashboard ringkasan untuk pimpinan, biasanya berisi KPI strategis dan indikator utama organisasi.
Fact Table	Tabel pusat dalam model dimensional yang berisi kejadian bisnis dan ukuran numerik, seperti transaksi, biaya, atau progres.
Failover Drill	Latihan memindahkan operasi ke sistem cadangan untuk menguji kesiapan *disaster recovery*.
FinOps	Praktik pengendalian biaya teknologi secara lintas peran agar biaya selaras dengan nilai bisnis.

Foreign Key	Kolom yang menghubungkan tabel fakta dengan tabel dimensi atau tabel lain.
Full Backup	Backup lengkap seluruh database atau seluruh dataset penting.
Full Extract	Proses mengambil seluruh data dari sumber, bukan hanya perubahan terbaru.
Governance	Lihat **Data Governance**.
Grain	Tingkat detail satu baris pada fact table, misalnya satu baris = satu item transaksi penjualan.
Green Data Warehouse	Pendekatan merancang DW agar hemat energi, hemat storage, hemat compute, dan minim pemborosan.
Historical Load	Proses memuat data masa lalu ke *Data Warehouse*, misalnya data transaksi lima tahun terakhir.
Index	Struktur tambahan pada database untuk mempercepat pencarian, filter, join, dan pengurutan data.
Incremental Backup	Backup yang hanya menyimpan perubahan sejak backup sebelumnya.
Incremental Extract	Proses mengambil hanya data baru atau data yang berubah sejak proses terakhir.
Initial Load	Proses pemuatan data pertama kali ke *Data Warehouse*.
Internal Data	Data yang hanya boleh digunakan di dalam organisasi.
KPI	*Key Performance Indicator*, indikator utama untuk mengukur kinerja, seperti penjualan, margin, SLA, atau progres proyek.
Least Privilege	Prinsip keamanan yang memberikan akses minimum sesuai kebutuhan kerja pengguna.
Logical Design	Desain logis yang menjelaskan entitas, relasi, fact, dimension, grain, dan aturan bisnis tanpa detail fisik database.
Maintenance	Aktivitas pemeliharaan rutin seperti vacuum, analyze, rebuild index, cleanup staging, archive, dan update statistik.
Materialized View	Hasil query yang disimpan secara fisik agar laporan atau dashboard berat dapat diakses lebih cepat.
Metadata	Data tentang data, seperti definisi kolom, sumber data, aturan transformasi, pemilik data, dan lineage.

Monitoring	Aktivitas memantau status ETL, query, storage, backup, performa, error, dan akses pengguna.
Near-real-time	Pemrosesan data dengan jeda sangat pendek, biasanya untuk kebutuhan operasional yang cepat.
OLAP	*Online Analytical Processing*, pendekatan pemrosesan data untuk analitik, ringkasan, tren, dan pelaporan.
OLTP	*Online Transaction Processing*, sistem transaksi operasional seperti penjualan, pembayaran, pemesanan, atau input harian.
OPEX	*Operational expenditure*, biaya operasional berjalan seperti cloud, listrik, bandwidth, support, SDM, dan monitoring.
Operational Manual	Dokumen panduan operasi yang menjelaskan cara menjalankan ETL, membaca log, menangani error, backup, dan eskalasi.
Partitioning	Teknik membagi tabel besar menjadi bagian lebih kecil, biasanya berdasarkan tanggal, agar query dan maintenance lebih efisien.
Personal Data	Data yang terkait dengan individu, seperti nama, nomor identitas, kontak, alamat, atau data keuangan pribadi.
Physical Design	Desain fisik database, termasuk tabel, kolom, tipe data, indeks, partisi, constraint, dan storage.
Pilot Unit	Unit terbatas yang digunakan untuk menguji sistem sebelum diperluas ke seluruh organisasi.
Platform Engineer	Peran yang mengelola platform, infrastruktur, database, monitoring, backup, dan reliabilitas sistem.
Primary Key	Kolom atau kombinasi kolom yang menjadi identitas unik suatu baris dalam tabel.
Production Environment	Lingkungan resmi yang digunakan pengguna organisasi untuk laporan, dashboard, dan analitik.
Project Charter	Dokumen awal proyek yang berisi tujuan, sponsor, scope, output, risiko, timeline, dan kriteria sukses.
Query Response Time	Waktu yang dibutuhkan sistem untuk menjawab query.
Queue Time	Waktu tunggu query sebelum diproses karena keterbatasan sumber daya atau antrean workload.
RACI	Matriks tanggung jawab: Responsible, Accountable, Consulted, dan Informed.

Raw Data	Data mentah dari sumber sebelum dibersihkan, ditransformasi, atau divalidasi.
RBAC	*Role-Based Access Control*, pengaturan akses berdasarkan peran pengguna.
Rebuild Index	Proses membangun ulang indeks agar struktur indeks lebih optimal.
Recovery Point Objective	RPO, batas maksimum kehilangan data yang masih dapat diterima setelah gangguan.
Recovery Time Objective	RTO, target waktu maksimal untuk memulihkan sistem setelah gangguan.
Referential Integrity	Kepastian bahwa relasi antar tabel valid, misalnya fact table memiliki dimensi yang sesuai.
Refresh Frequency	Frekuensi pembaruan data, misalnya harian, mingguan, bulanan, per jam, atau near-real-time.
Restricted Data	Data sangat sensitif yang aksesnya harus sangat dibatasi, seperti kredensial, audit kritikal, atau data personal sensitif.
Retention	Kebijakan berapa lama data disimpan secara aktif atau diarsipkan.
Rollback Plan	Rencana kembali ke kondisi aman jika deployment gagal atau terjadi masalah besar.
RPO	Lihat **Recovery Point Objective**.
RTO	Lihat **Recovery Time Objective**.
Schema	Struktur logis database yang berisi kumpulan tabel, view, fungsi, dan objek database lainnya.
Security Reviewer	Peran yang meninjau akses, masking, audit log, klasifikasi data, dan kepatuhan keamanan.
Segregation of Duties	Pemisahan tugas kritis agar tidak satu orang menguasai seluruh proses tanpa kontrol.
Slowly Changing Dimension	Teknik mengelola perubahan data dimensi dari waktu ke waktu, seperti perubahan alamat pelanggan.
Snapshot	Salinan kondisi sistem pada titik waktu tertentu, biasanya sebelum perubahan besar atau deployment.
Snowflake Schema	Model dimensional dengan tabel dimensi yang dinormalisasi menjadi beberapa tabel turunan.

Source System	Sistem asal data, seperti aplikasi transaksi, file, API, log, atau sistem operasional lainnya.
Source-to-Target Mapping	Pemetaan dari kolom sumber ke kolom target beserta aturan transformasinya.
Staging Area	Area sementara untuk menyimpan data sebelum dibersihkan, ditransformasi, dan dimuat ke warehouse.
Star Schema	Model dimensional dengan satu fact table di tengah dan beberapa dimension table di sekelilingnya.
Storage Growth	Pertumbuhan kapasitas data dari waktu ke waktu.
Surrogate Key	Kunci buatan di tabel dimensi yang tidak bergantung langsung pada kode dari sistem sumber.
Sustainability	Prinsip keberlanjutan dalam DW, mencakup efisiensi energi, storage, compute, biaya, dan retensi.
Testing Environment	Lingkungan untuk menguji ETL, kualitas data, performa, akses, dan error handling sebelum staging atau production.
TCO	*Total Cost of Ownership*, total biaya kepemilikan sistem selama periode tertentu.
UAT	*User Acceptance Testing*, pengujian oleh pengguna bisnis untuk memastikan sistem sesuai kebutuhan dan siap digunakan.
Vacuum	Proses maintenance database untuk membersihkan ruang dan menjaga kesehatan tabel, terutama pada PostgreSQL.
View	Objek database berupa query tersimpan yang menampilkan data dari satu atau lebih tabel.
Workload	Beban kerja sistem, seperti ETL, dashboard, query analis, backup, atau refresh materialized view.
Workload Management	Pengelolaan beban kerja agar ETL, dashboard, dan query berat tidak saling mengganggu.

Referensi

Apache Superset. (n.d.). *Exploring data*. Apache Superset Documentation. https://superset.apache.org/user-docs/using-superset/exploring-data/

Avelar, V., Azevedo, D., & French, A. (2012). *PUE™: A comprehensive examination of the metric*. The Green Grid. https://datacenters.lbl.gov/sites/default/files/WP49-PUE%20A%20Comprehensive%20Examination%20of%20the%20Metric_v6.pdf

Badan Pemeriksa Keuangan Republik Indonesia. (2022). *Undang-Undang Nomor 27 Tahun 2022 tentang Pelindungan Data Pribadi*. https://peraturan.bpk.go.id/Details/229798/uu-no-27-tahun-2022

Bharathan, G. (2024). *Ultimate Snowflake Architecture for Cloud Data Warehousing*.

DAMA International. (2024). *The DAMA guide to the data management body of knowledge (DAMA-DMBOK2R)*. Technics Publications. https://dama.org/dmbok2r-infographics/

DAMA International. (n.d.). *DAMA-DMBOK: Data management body of knowledge*. https://dama.org/learning-resources/dama-data-management-body-of-knowledge-dmbok/

DAMA International. (n.d.). *What is data management?* https://dama.org/about-dama/what-is-data-management/

Debezium. (n.d.). *Debezium documentation*. https://debezium.io/documentation/

Debezium. (n.d.). *Debezium reference documentation*. https://debezium.io/documentation/reference/stable/index.html

Debezium. (n.d.). *Debezium: Stream changes from your database*. https://debezium.io/

Deborah, L. (2025). *Data Mining and Warehousing Techniques with Machine Learning Concepts*.

EDM Council. (n.d.). *DCAM framework*. https://edmcouncil.org/frameworks/

FinOps Foundation. (n.d.). *FinOps framework*. https://www.finops.org/framework/

FinOps Foundation. (n.d.). *Optimize usage & cost*. https://www.finops.org/framework/domains/optimize-usage-cost/

FinOps Foundation. (n.d.). *Usage optimization*. https://www.finops.org/framework/capabilities/usage-optimization/

Green Software Foundation. (n.d.). *Software Carbon Intensity specification*. https://sci.greensoftware.foundation/

Green Software Foundation. (n.d.). *Software Carbon Intensity: SCI standard*. https://greensoftware.foundation/standards/sci/

Hermans, K. (2023). *Mastering Data Warehousing: A Comprehensive Guide*.

IBM. (n.d.). *ELT vs. ETL: Similarities and differences*. https://www.ibm.com/think/topics/elt-vs-etl

IBM. (n.d.). *What is ETL?* https://www.ibm.com/think/topics/etl

Karamagi, R. (2020). *Data Mining and Data Warehouse*.

Kimball Group. (n.d.). *Dimensional modeling techniques*. https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/

Kimball Group. (2008). *Slowly changing dimensions*. https://www.kimballgroup.com/2008/08/slowly-changing-dimensions/

Kumar, J. (2024). *Data Warehouse and Data Mining: Concepts, Techniques, and Applications*.

Metabase. (n.d.). *Questions*. Metabase Documentation. https://www.metabase.com/docs/latest/questions/introduction

Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing* (NIST Special Publication 800-145). National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf

National Institute of Standards and Technology. (n.d.). *Least privilege*. NIST Computer Security Resource Center Glossary. https://csrc.nist.gov/glossary/term/least_privilege

Open Worldwide Application Security Project. (n.d.). *Logging cheat sheet*. OWASP Cheat Sheet Series. https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

Open Worldwide Application Security Project. (2021). *A09:2021 – Security logging and monitoring failures*. OWASP Top 10. https://owasp.org/Top10/2021/A09_2021-Security_Logging_and_Monitoring_Failures/

PostgreSQL Global Development Group. (n.d.). *COPY*. PostgreSQL Documentation. https://www.postgresql.org/docs/current/sql-copy.html

PostgreSQL Global Development Group. (n.d.). *EXPLAIN*. PostgreSQL Documentation. https://www.postgresql.org/docs/current/sql-explain.html

PostgreSQL Global Development Group. (n.d.). *Materialized views*. PostgreSQL Documentation. https://www.postgresql.org/docs/current/rules-materializedviews.html

PostgreSQL Global Development Group. (n.d.). *Monitoring database activity*. PostgreSQL Documentation. https://www.postgresql.org/docs/current/monitoring.html

PostgreSQL Global Development Group. (n.d.). *Partitioning*. PostgreSQL Documentation. https://www.postgresql.org/docs/current/ddl-partitioning.html

PostgreSQL Global Development Group. (n.d.). *Routine vacuuming*. PostgreSQL Documentation. https://www.postgresql.org/docs/current/routine-vacuuming.html

PostgreSQL Global Development Group. (n.d.). *SQL dump, file system level backup, and continuous archiving*. PostgreSQL Documentation. https://www.postgresql.org/docs/9.1/backup.html

PostgreSQL Global Development Group. (n.d.). *Continuous archiving and point-in-time recovery*. PostgreSQL Documentation. https://www.postgresql.org/docs/current/continuous-archiving.html

PostgreSQL Global Development Group. (n.d.). *VACUUM*. PostgreSQL Documentation. https://www.postgresql.org/docs/current/sql-vacuum.html

SFIA Foundation. (n.d.). *Data engineering*. SFIA 9. https://sfia-online.org/en/sfia-9/skills/data-engineering

SFIA Foundation. (n.d.). *Data management*. SFIA 9. https://sfia-online.org/en/sfia-9/skills/data-management

SFIA Foundation. (n.d.). *Database administration*. SFIA 9. https://sfia-online.org/en/sfia-9/skills/database-administration

Snowflake. (n.d.). *Multi-cluster warehouses*. Snowflake Documentation. https://docs.snowflake.com/en/user-guide/warehouses-multicluster

Snowflake. (n.d.). *Virtual warehouses*. Snowflake Documentation. https://docs.snowflake.com/en/user-guide/warehouses-overview

Snowflake. (n.d.). *Warehouse load monitoring*. Snowflake Documentation. https://docs.snowflake.com/en/user-guide/warehouses-load-monitoring

U.S. Department of Energy, Federal Energy Management Program, & National Renewable Energy Laboratory. (2024). *Best practices guide for energy-efficient data center design*. https://www.energy.gov/sites/default/files/2024-07/best-practice-guide-data-center-design.pdf

Tentang Penulis



Onno W. Purbo, saat ini bertugas sebagai rektor di Institut Teknologi Tangerang Selatan (ITTS). Onno memperoleh gelar Ph.D bidang Electrical Engineering dari University of Waterloo, Canada, adalah seorang copyleft, educator dan ICT evangelist. Dia sudah mempublikasikan 50++ buku, termasuk free ICT ebook untuk sekolah tahun 2008. Beberapa buku terakhirnya adalah "Internet-TCP/IP: Konsep Dan Implementasi", 2018; "Sistem Operasi, Konsep Dan Membuat Linux OpenWRT Dan ROM Android", 2019; "IPv6 Untuk Mendukung Operasi Jaringan Dan Domain Name System", 2019; "Kubernetes untuk Pemula", 2024 dan "Membuat Operator Seluler 5G Sendiri", 2024.

Dia memimpin sambungan pertama Internet di Institut Teknologi Bandung, tahun 1993-2000, dan menggunakannya untuk membuat jaringan Internet pendidikan yang pertama di Indonesia. Dia membebaskan frekuensi WiFi, memperkenalkan RT/RW-net, antenna Wajanbolic dan jaringan selular OpenBTS dan private 5G sendiri. Dia memimpin jaringan telepon pertama di atas Internet, VoIP Merdeka, yang kemudian hari dikenal sebagai VoIP Rakyat berbasis SIP dan menggunakan kode area +62520 dan +62521. Dia saat ini aktif memperkenalkan e-Learning, dan menjalankan server e-Learning di http://lms.onnocenter.or.id/moodle/ 66,000++ siswa / mahasiswa dan https://opencourse.itts.ac.id 39.000+ mahasiswa secara gratis. Email: onno@indo.net.id