JSON untuk Interaksi Database MySQL

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

Institut Teknologi Tangerang Selatan (ITTS)
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

JSON bukan pengganti desain database yang baik; JSON adalah jembatan fleksibel agar database relasional mampu berbicara dengan API, aplikasi modern, otomasi workflow, dan data semi-terstruktur tanpa kehilangan disiplin desain.

Kata Pengantar

Puji syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa atas tersusunnya buku **JSON untuk Interaksi Database MySQL** ini. Buku ini disusun untuk membantu mahasiswa memahami bagaimana **JSON, MySQL, Python, dan n8n** dapat bekerja bersama dalam membangun pipeline data modern yang praktis, terstruktur, dan aman.

Buku ini tidak hanya membahas teori, tetapi juga menekankan praktik langsung: mulai dari memahami struktur JSON, menyimpan data ke MySQL, melakukan *query*, validasi dengan **JSON Schema**, otomasi dengan n8n, hingga membangun proyek akhir berbasis pipeline nyata. Pendekatan yang digunakan adalah **open source, praktis, dan bertahap**, agar mahasiswa tidak hanya mampu mengikuti contoh, tetapi juga memahami alasan teknis di balik setiap keputusan desain.

Pesan utama buku ini sederhana: **JSON memberi fleksibilitas, tetapi fleksibilitas tanpa validasi, desain skema, keamanan, dan dokumentasi akan menjadi sumber masalah baru**. Karena itu, buku ini menempatkan desain database hybrid, validasi data, prepared statement, indexing, backup, audit trail, dan etika penggunaan data sebagai bagian penting dari keterampilan teknis mahasiswa.

Semoga buku ini dapat menjadi bekal awal bagi mahasiswa untuk membangun sistem data yang lebih rapi, aman, terukur, dan siap dikembangkan di dunia akademik maupun profesional.

Jakarta, Mei 2026
Onno W. Purbo

Executive Summary

Buku **JSON untuk Interaksi Database MySQL** menempatkan JSON bukan sebagai tren teknis sesaat, tetapi sebagai **fondasi penting integrasi data modern** antara aplikasi, API, workflow, dan database relasional. Mahasiswa diajak memahami bahwa MySQL tidak hanya menyimpan tabel klasik, tetapi juga mampu menangani data semi-terstruktur melalui kolom JSON, fungsi query, validasi, indexing, dan integrasi dengan Python serta n8n. Namun pesan utama buku ini tajam: **fleksibilitas tanpa desain akan berubah menjadi kekacauan data**. Karena itu, buku ini membangun alur belajar yang strategis dari dasar struktur JSON, CRUD, Python, SQLAlchemy, query canggih, `JSON_TABLE`, desain skema hybrid, optimasi performa, JSON Schema, otomasi n8n, hingga proyek akhir end-to-end. Pendekatan yang digunakan bersifat **hands-on**, open source, dan relevan dengan kebutuhan dunia kerja: membaca file JSON, validasi data, prepared statement, batch insert, export laporan, workflow automation, audit trail, backup, dan dokumentasi teknis. Buku ini juga menekankan bahwa JSON valid belum tentu data benar; kualitas data harus dijaga melalui **JSON Schema, validasi sebelum insert, privilege terbatas, logging, masking data sensitif, dan pengujian query**. Di tingkat strategis, mahasiswa tidak hanya belajar menulis query, tetapi belajar merancang pipeline data yang aman, dapat diaudit, bisa dikembangkan, dan siap dipresentasikan sebagai portofolio profesional. Proyek akhir menjadi puncak pembelajaran: sistem kecil yang nyata dengan database hybrid, Python, n8n, laporan otomatis, optimasi query, dan keamanan dasar. Buku ini menginspirasi mahasiswa untuk naik kelas dari sekadar pengguna database menjadi **perancang pipeline data modern** yang kritis, disiplin, dan bertanggung jawab.

Daftar Isi

Kata Pengantar **4**
Executive Summary **5**
Daftar Isi **6**
Bab 1 — Mengapa JSON Penting untuk Interaksi Database Modern? **16**
Tujuan Pembelajaran 16
1.1 Mengapa JSON Menjadi Penting? 17
1.2 Evolusi Data: Dari Tabel Relasional ke Data Semi-Terstruktur 18
1.3 JSON sebagai Bahasa Pertukaran Data Modern 19
1.4 Mengapa JSON Lebih Mudah Dibaca Dibanding Format yang Terlalu Verbose? 20
1.5 Posisi MySQL dalam Aplikasi API, Backend, Dashboard, dan Otomasi 21
1.6 Peran Python sebagai Scripting Engine 22
1.7 Peran n8n sebagai Workflow Automation Engine 23
1.8 Kapan Memakai Kolom Relasional, Kapan Memakai Kolom JSON? 24
1.9 Risiko Awal: Fleksibel Bukan Berarti Bebas 26
Hands-on Bab 1 ..... **27**
Praktik 1 — Instalasi Ringkas MySQL Community Server 27
Praktik 2 — Membuat User dan Database Latihan 28
Praktik 3 — Membuat Tabel Pertama dengan Kolom JSON 29
Praktik 4 — Insert Satu Dokumen JSON Sederhana 30
Praktik 5 — Query Pertama Menggunakan JSON_EXTRACT 31
Praktik 6 — Instalasi Ringkas Python dan Pustaka Database 32
Praktik 7 — Instalasi Ringkas n8n secara Lokal 33
Mini-Lab Bab 1 — Database Profil Mahasiswa Berbasis JSON **34**
Tujuan 34
Langkah 34
Kesalahan Umum Mahasiswa **35**
Checklist Akhir Bab **35**
Ringkasan Bab **36**
Bab 2 — Dasar JSON di MySQL: Struktur, Validasi, dan JSON Path **37**
Tujuan Pembelajaran 37
2.1 JSON Bukan Sekadar Teks Panjang 37
2.2 Struktur Dasar JSON 38
2.3 Object, Array, dan Key-Value 40
2.4 Perbedaan JSON, TEXT, VARCHAR, dan BLOB 41
2.5 Validasi Otomatis JSON di MySQL 42
2.6 Fungsi Dasar JSON di MySQL 44
2.6.1 JSON_VALID 44
2.6.2 JSON_TYPE 45
2.6.3 JSON_EXTRACT 46
2.7 Operator -> dan ->> 47
2.8 Memahami JSON Path 48
Hands-on Bab 2 ..... **49**

2.9 Membuat Database dan Tabel mahasiswa_profile....49
2.10 Insert Profil Mahasiswa dalam Format JSON.... 50
2.11 Query Nama, Alamat, Minat, dan Skill....52
2.12 Filter Mahasiswa Berdasarkan Isi JSON....53
2.13 Mendeteksi JSON Valid dan Invalid.... 55
2.14 Latihan Memperbaiki JSON Rusak.... 56
Kasus 1 — Koma Berlebih.... 56
Kasus 2 — Key Tidak Memakai Double Quote.... 56
Kasus 3 — String Tidak Memakai Double Quote....57
Kasus 4 — Key Tidak Konsisten.... 57
2.15 Validasi JSON dengan Python.... 58
Mini-Lab Bab 2 — Profil Mahasiswa Berbasis JSON....59
Tujuan.... 59
Dataset Mini.... 59
Tugas Praktik.... 60
Query Jawaban Utama....61
Kesalahan Umum Mahasiswa.... 62
Checklist Akhir Bab.... 62
Ringkasan Bab.... 63
Bab 3 — CRUD JSON dengan MySQL: Insert, Read, Update, Delete....64
Tujuan Pembelajaran.... 64
3.1 Pola CRUD untuk Data JSON....65
3.2 Mengapa CRUD JSON Berbeda dari CRUD Tabel Biasa?....66
3.3 Desain Tabel Hybrid: Relasional + JSON....67
3.4 Insert Dokumen JSON.... 68
3.5 Read Field JSON Tertentu.... 69
3.6 Update Sebagian Isi JSON dengan JSON_SET....70
3.7 Update Nested Object....71
3.8 Menambahkan Array dan Object Baru....72
3.9 Menghapus Elemen JSON dengan JSON_REMOVE....73
3.10 Delete Row vs Delete Item JSON....74
3.11 Delete Item Berdasarkan Kondisi Isi JSON....75
3.12 Transaction Sederhana untuk Operasi Penting.... 76
3.13 Kapan JSON Membuat Sistem Lebih Fleksibel?.... 77
3.14 Kapan JSON Merusak Desain Database?....78
Hands-on Bab 3....79
3.15 Membuat Tabel Produk.... 79
3.16 Insert Produk Elektronik dengan Spesifikasi Fleksibel....80
3.17 Read Field JSON Produk....82
3.18 Update Stok, Warna, RAM, Ukuran Layar, dan Garansi....83
3.19 Menambahkan Atribut Baru Tanpa Mengubah Struktur Tabel....84
3.20 Menghapus Elemen JSON....85
3.21 Delete Item Berdasarkan Kondisi Isi JSON....86
3.22 Praktik Transaction Aman.... 87

Mini-Lab Bab 3 — Katalog Produk Sederhana Berbasis MySQL JSON.....88
Tujuan..... 88
Tugas..... 88
Contoh Produk Tambahan.....88
Query Laporan Mini.....89
Kesalahan Umum Mahasiswa..... 90
Checklist Akhir Bab..... 91
Ringkasan Bab..... 91
Bab 4 — Python untuk Interaksi MySQL JSON.....92
Tujuan Pembelajaran..... 92
4.1 Mengapa Python Penting untuk MySQL JSON?.....93
4.2 Alur Kerja Python + JSON + MySQL.....94
4.3 Instalasi mysql-connector-python.....95
4.4 Struktur Dasar Script Database.....96
4.5 Cursor Biasa dan Cursor Dictionary.....97
4.6 Prepared Statement untuk Mengurangi Risiko SQL Injection.....98
4.7 Konversi Python dict dan list ke JSON.....99
4.8 Menyiapkan Tabel mahasiswa_profile.....100
4.9 Membuat File students.json..... 101
4.10 Membaca File JSON dengan Python.....102
4.11 Insert Banyak Data JSON ke MySQL.....103
4.12 Batch Insert dengan executemany.....105
4.13 Query Field JSON dari Python.....107
4.14 Query dengan Parameter dari Python.....108
4.15 Export Hasil Query Menjadi result.json..... 109
4.16 Menangani Error Koneksi..... 111
4.17 Menangani Error SQL.....112
4.18 Menangani JSON Invalid.....113
4.19 Validasi Struktur Dasar Sebelum Insert.....114
4.20 Transaction dan Rollback Sederhana.....115
Hands-on Utama Bab 4 — Importer JSON Mahasiswa..... 116
4.21 Struktur Folder Praktik.....116
4.22 Script Lengkap importer.py.....117
4.23 Penjelasan Script importer.py.....119
4.24 Query dari Python dan Export JSON..... 121
Mini-Lab Bab 4 — Importer JSON Mahasiswa.....123
Tujuan.....123
Tugas Praktik..... 123
Output yang Diharapkan..... 124
Kesalahan Umum Mahasiswa..... 124
Checklist Akhir Bab..... 125
Ringkasan Bab..... 125
Bab 5 — SQLAlchemy: Interaksi Database yang Lebih Rapi dan Terstruktur..... 126
Tujuan Pembelajaran..... 126

5.1 Mengapa SQLAlchemy Penting? 127
5.2 Perbedaan SQL Mentah dan ORM 128
5.3 Engine, Connection, Session, dan Model 129
Engine 129
Connection 129
Session 129
Model 129
5.4 Instalasi SQLAlchemy 130
5.5 Membuat Koneksi SQLAlchemy 130
5.6 Membuat Model UserProfile 131
5.7 Membuat Tabel dari Model 132
5.8 Insert Object Python ke Kolom JSON 133
5.9 Insert Banyak User 134
5.10 Query Data Dasar dengan SQLAlchemy 135
5.11 Query Berdasarkan Field JSON 136
5.12 Update Nested JSON 138
5.13 Update JSON dengan SQL Mentah melalui SQLAlchemy 139
5.14 Export Hasil Query ke Format JSON API-style 140
5.15 Membandingkan SQL Mentah dan SQLAlchemy 142
SQL mentah dengan connector 142
SQLAlchemy ORM 142
5.16 Kapan Memakai mysql-connector-python? 143
5.17 Kapan Memakai SQLAlchemy? 143
5.18 Risiko ORM: Jangan Sampai Tidak Paham SQL 144
Hands-on Bab 5 145
5.19 Struktur Folder Praktik 145
5.20 File db.py 145
5.21 File models.py 146
5.22 File create_tables.py 146
5.23 File insert_user.py 147
5.24 File query_by_json.py 148
5.25 File update_nested_json.py 148
5.26 File export_api_style.py 149
Mini-Lab Bab 5 — Backend Data Profil Sederhana 150
Tujuan 150
Tugas Praktik 150
Contoh Output API-style 151
Kesalahan Umum Mahasiswa 151
Checklist Akhir Bab 152
Ringkasan Bab 152
Bab 6 — Query JSON Canggih: Filtering, Aggregation, dan JSON_TABLE 153
Tujuan Pembelajaran 153
6.1 Mengapa Query JSON Canggih Penting? 154
6.2 Dataset Order Berbasis JSON 154

6.3 Membuat Tabel customers..... 155
6.4 Membuat Tabel orders_json..... 156
6.5 Insert Dataset Customer..... 157
6.6 Insert Dataset Order JSON..... 157
6.7 Query Nested JSON..... 160
6.8 Filter Berdasarkan Object..... 161
6.9 Filter Berdasarkan Array dengan JSON_CONTAINS..... 162
6.10 Filter dengan JSON_OVERLAPS..... 163
6.11 Filter dengan MEMBER OF..... 164
6.12 Mengubah JSON Menjadi Tabel dengan JSON_TABLE..... 165
6.13 Query Produk dalam Array Order..... 166
6.14 Join Tabel Relasional dengan Data JSON..... 167
6.15 Filter Customer Berdasarkan Kota..... 168
6.16 Membuat Laporan Total Order per Kota..... 169
6.17 Laporan Produk Terlaris..... 170
6.18 Aggregation dengan JSON_ARRAYAGG..... 171
6.19 Aggregation dengan JSON_OBJECTAGG..... 172
6.20 Membuat Laporan JSON Lengkap..... 173
6.21 Batasan Performa Query JSON..... 174
Hands-on Bab 6..... 175
6.22 Membuat View Item Order..... 175
6.23 Laporan Total per Kota dengan View..... 176
6.24 Laporan Produk Terlaris dengan View..... 176
6.25 Export Laporan dengan Python..... 176
Mini-Lab Bab 6 — Laporan Transaksi dari Data JSON..... 178
Tujuan..... 178
Tugas Praktik..... 178
Output yang Diharapkan..... 179
Kesalahan Umum Mahasiswa..... 179
Checklist Akhir Bab..... 180
Ringkasan Bab..... 180
Bab 7 — Indexing, Performance, dan Desain Skema Hybrid..... 181
Tujuan Pembelajaran..... 181
7.1 Mengapa Query JSON Bisa Lambat?..... 181
7.2 Kolom JSON Tidak Diindeks Langsung..... 182
7.3 Desain Skema Hybrid: Kunci Performa JSON..... 183
7.4 Generated Column untuk Field JSON Penting..... 184
7.5 Virtual vs Stored Generated Column..... 185
7.6 Membuat Tabel Profil Mahasiswa untuk Optimasi..... 186
7.7 Insert Dataset Latihan..... 187
7.8 Membandingkan Query Tanpa dan Dengan Generated Column..... 188
7.9 Membaca Hasil EXPLAIN..... 189
7.10 Studi Kasus: Mencari User Berdasarkan Nomor HP dalam JSON..... 190
7.11 Studi Kasus: Mencari Mahasiswa Berdasarkan Kota..... 191

7.12 Multi-Valued Index untuk Array JSON....192
7.13 MEMBER OF, JSON_CONTAINS, dan JSON_OVERLAPS dengan MVI....194
MEMBER OF....194
JSON_CONTAINS....194
JSON_OVERLAPS....194
7.14 Dampak Index terhadap Insert dan Update....195
7.15 Normalisasi vs Fleksibilitas....196
Hands-on Bab 7....197
7.16 Membandingkan Query Sebelum dan Sesudah Index....197
Query langsung dari JSON....197
Query memakai generated column....197
7.17 Membuat Index Setelah Tabel Ada....198
7.18 Studi Kasus Produk Berdasarkan Tag dalam Array....199
7.19 Export Hasil Optimasi dengan Python....201
Mini-Lab Bab 7 — Optimasi Tabel Profil Mahasiswa....203
Tujuan....203
Tugas Praktik....203
Output yang Diharapkan....204
Kesalahan Umum Mahasiswa....204
Checklist Akhir Bab....205
Ringkasan Bab....205
Bab 8 — Validasi JSON dengan JSON Schema Sebelum Masuk Database....206
Tujuan Pembelajaran....206
8.1 Mengapa JSON Valid Belum Tentu Benar?....207
8.2 Konsep JSON Schema....208
8.3 JSON Schema sebagai Pengurang Ambiguitas....209
8.4 Mengenal $schema dan Dialect....210
8.5 Keyword Penting JSON Schema....211
8.6 Validasi Object....212
8.7 Validasi Array....213
8.8 Validasi Email, Tanggal, Angka, dan String....215
8.9 Posisi JSON Schema dalam Pipeline....216
Hands-on Bab 8....217
8.10 Instalasi Pustaka Python....217
8.11 Membuat student_schema.json....217
8.12 Membuat students.json....219
8.13 Validasi File students.json dengan Python....220
8.14 Menolak Field Wajib yang Kosong....221
8.15 Menolak Email Invalid....222
8.16 Menolak Tipe Data yang Salah....223
8.17 Simpan Hanya JSON yang Lolos Validasi ke MySQL....224
8.18 Laporan Error yang Bisa Dibaca....227
8.19 Validasi Tambahan di MySQL....228
8.20 Posisi JSON Schema dalam n8n....229

Mini-Lab Bab 8 — Gerbang Validasi Data Pendaftaran Mahasiswa..... 230
Tujuan..... 230
Struktur Folder.....230
Tugas Praktik..... 230
Query Cek Data Valid.....231
Script Export Data Valid..... 231
Kesalahan Umum Mahasiswa..... 232
Checklist Akhir Bab..... 233
Ringkasan Bab..... 233
Bab 9 — n8n untuk Automasi JSON dan MySQL.....234
Tujuan Pembelajaran..... 234
9.1 Mengapa n8n Penting untuk Mahasiswa?..... 235
9.2 Konsep Workflow Automation..... 236
9.3 Struktur Data n8n: Item, Field, dan JSON.....237
9.4 Node Penting dalam Bab Ini.....238
Webhook..... 238
Edit Fields / Set..... 238
Code.....238
HTTP Request.....238
MySQL..... 239
9.5 Alur Utama: Terima JSON → Validasi → Bersihkan → Simpan ke MySQL.....240
9.6 Menyiapkan Tabel MySQL..... 241
9.7 Membuat Webhook Penerima JSON..... 242
9.8 Mengirim Data dari curl..... 243
9.9 Mengirim Data dari Hoppscotch.....244
9.10 Membersihkan Field dengan Edit Fields / Set.....245
9.11 Validasi Sederhana dengan Code Node..... 246
9.12 Memisahkan Data Valid dan Invalid..... 248
9.13 Insert JSON ke MySQL dari n8n.....249
9.14 Query MySQL lalu Kirim Response JSON.....250
9.15 Update Data dari n8n..... 251
9.16 Error Handling Sederhana.....252
9.17 Export dan Import Workflow JSON..... 253
9.18 Kapan Logic Dikerjakan di n8n?.....254
9.19 Kapan Logic Dikerjakan di Python?..... 255
Hands-on Bab 9 — API Pendaftaran Mahasiswa Sederhana..... 256
9.20 Desain Workflow..... 256
9.21 Payload Valid.....256
9.22 Payload Invalid.....257
9.23 Query Cek Data di MySQL.....258
9.24 Export Data dari MySQL dengan Python..... 259
Mini-Lab Bab 9 — API Pendaftaran Mahasiswa Tanpa Framework Berat.....261
Tujuan..... 261
Tugas Praktik..... 261

Output yang Diharapkan..... 262
Kesalahan Umum Mahasiswa..... 262
Checklist Akhir Bab..... 263
Ringkasan Bab..... 263
Bab 10 — Proyek Akhir: Pipeline JSON, Python, n8n, MySQL, Security.....264
Tujuan Pembelajaran..... 264
10.1 Posisi Bab 10 dalam Buku..... 264
10.2 Arsitektur Integrasi Python + n8n + MySQL.....265
10.3 Pilihan Proyek Akhir..... 266
1. Sistem Pendaftaran Mahasiswa Berbasis JSON + MySQL + n8n..... 266
2. Inventory Sederhana dengan Spesifikasi Produk Fleksibel..... 266
3. Log Collector JSON untuk Monitoring Aplikasi.....266
4. Mini ETL Pipeline dari API Publik ke MySQL..... 267
5. Dashboard Reporting Berbasis Query JSON.....267
6. Sistem Import Data UMKM dari File JSON ke MySQL..... 267
7. Sistem Validasi Data Form Menggunakan JSON Schema + Python..... 267
10.4 Arsitektur Proyek yang Direkomendasikan..... 268
10.5 Desain Database Hybrid..... 269
10.6 Tabel Logging dan Audit Trail Sederhana..... 270
10.7 JSON Schema Proyek Akhir..... 271
10.8 Struktur Folder Proyek..... 273
10.9 Script Python Validasi.....274
10.10 Script Python Import ke MySQL.....276
10.11 Workflow n8n Proyek Akhir..... 278
10.12 Query Laporan dari MySQL..... 280
Laporan jumlah mahasiswa per prodi..... 280
Laporan mahasiswa per kota dari JSON.....280
Laporan skill dalam JSON dengan JSON_TABLE..... 280
10.13 Export Laporan dengan Python.....281
10.14 Optimasi Satu Query dengan Generated Column.....283
10.15 User MySQL dengan Privilege Terbatas..... 284
10.16 Prepared Statement dan Pencegahan SQL Injection..... 285
10.17 Backup dan Restore Sederhana..... 285
10.18 Versioning Struktur JSON..... 286
10.19 Etika Penggunaan Data..... 287
10.20 AI-assisted Query: Tren Masa Depan, Tetap Harus Diverifikasi..... 288
Hands-on Akhir Bab 10.....289
10.21 Langkah 1 — Buat Database dan Tabel.....289
10.22 Langkah 2 — Buat User Aplikasi.....290
10.23 Langkah 3 — Buat Data Contoh..... 291
10.24 Langkah 4 — Jalankan Validasi dan Import.....292
10.25 Langkah 5 — Buat Workflow n8n.....293
10.26 Langkah 6 — Buat Query Laporan.....294
10.27 Langkah 7 — Backup Database.....295

10.28 Langkah 8 — Dokumentasi Teknis....296
10.29 Checklist Keamanan Proyek....297
10.30 Rubrik Penilaian Proyek Akhir....297
10.31 Format Presentasi Hasil Proyek....298
Mini-Lab Bab 10 — Proyek Akhir End-to-End....299
Tujuan....299
Output Wajib....299
Kriteria Lulus Minimal....299
Kesalahan Umum Mahasiswa....300
Checklist Akhir Bab....300
Ringkasan Bab....301
Lampiran A — Cheat Sheet JSON MySQL....302
A.1 Fungsi Validasi dan Tipe Data....302
A.2 Fungsi Membaca JSON....303
A.3 Fungsi Mengubah JSON....304
A.4 Fungsi Pencarian JSON....305
A.5 Fungsi Agregasi JSON....306
A.6 Mengubah JSON Menjadi Tabel dengan JSON_TABLE....307
Lampiran B — Template Database Praktik....308
B.1 Tabel mahasiswa_profile....308
B.2 Tabel produk....308
B.3 Tabel orders_json....309
B.4 Tabel api_logs....309
B.5 Tabel workflow_events....310
B.6 Tabel validated_students....310
Lampiran C — Contoh File JSON Latihan....311
C.1 Dataset Mahasiswa....311
C.2 Dataset Produk....312
C.3 Dataset Transaksi....313
C.4 Dataset Log Aplikasi....314
C.5 Dataset Response API....315
C.6 Dataset UMKM Sederhana....316
Lampiran D — Script Python Lengkap....317
D.1 Koneksi MySQL....317
D.2 Import JSON ke MySQL....318
D.3 Validasi JSON Schema....320
D.4 Export JSON dari MySQL....321
D.5 Batch Insert....322
D.6 Error Handling dan Transaction....323
D.7 SQLAlchemy Model....324
Lampiran E — Workflow n8n JSON....325
E.1 Workflow Webhook ke MySQL....325
E.2 Workflow API ke MySQL....326
E.3 Workflow Transform JSON....326

E.4 Workflow Validasi Sederhana..... 327
E.5 Workflow Reporting Otomatis..... 327
E.6 Workflow Error Notification..... 328
Lampiran F — Checklist Keamanan dan Validasi..... 329
F.1 Prepared Statement..... 329
F.2 User Privilege..... 329
F.3 JSON Validation..... 330
F.4 Backup.....330
F.5 Logging.....331
F.6 Masking Data Sensitif.....331
F.7 Audit Trail Sederhana.....332
Lampiran G — Rubrik Proyek Akhir Mahasiswa..... 333
G.1 Rubrik Penilaian..... 333
G.2 Level Penilaian..... 333
A — Sangat Baik.....333
B — Baik..... 334
C — Cukup.....334
D — Kurang.....334
E — Tidak Memadai.....334
G.3 Checklist Pengumpulan.....335
G.4 Format Presentasi 7 Slide..... 335
G.5 Pertanyaan Uji Saat Presentasi.....335
Glossary.....336
Referensi.....342
Tentang Penulis.....345

Bab 1 — Mengapa JSON Penting untuk Interaksi Database Modern?

> **JSON bukan pengganti desain database yang baik; JSON adalah jembatan fleksibel agar database relasional mampu berbicara dengan API, aplikasi modern, otomasi workflow, dan data semi-terstruktur tanpa kehilangan disiplin desain.**

Tujuan Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

- Menjelaskan mengapa **JSON penting** dalam aplikasi modern.
- Memahami hubungan antara **database relasional**, **API**, **dashboard**, dan **otomasi workflow**.
- Membedakan kapan data sebaiknya disimpan sebagai **kolom relasional biasa** dan kapan layak disimpan sebagai **kolom JSON**.
- Membuat database latihan dengan **MySQL Community Server**.
- Membuat tabel pertama dengan **kolom JSON**.
- Melakukan `INSERT` dokumen JSON sederhana.
- Mengambil isi JSON menggunakan `JSON_EXTRACT`.

Bab 1 diletakkan sebagai pembuka tentang pentingnya JSON untuk interaksi database modern, API, webhook, aplikasi, dashboard, log sistem, Python, n8n, dan risiko awal seperti data berantakan, validasi lemah, serta query lambat.

1.1 Mengapa JSON Menjadi Penting?

Aplikasi modern tidak hidup sendirian. Sebuah aplikasi kampus, toko daring, sistem absensi, dashboard monitoring, atau sistem log biasanya harus berbicara dengan banyak komponen lain. Data dikirim dari form, diterima oleh API, diproses oleh script, disimpan ke database, lalu ditampilkan kembali sebagai laporan.

Di tengah alur itu, **JSON menjadi bahasa pertukaran data**.

JSON dipakai karena bentuknya sederhana, mudah dibaca, mudah dikirim melalui API, dan mudah diproses oleh banyak bahasa pemrograman. Buku *MySQL & JSON* menjelaskan bahwa JSON populer karena mudah dibaca manusia, mudah diparse oleh program, dan tidak terlalu rumit dibanding format yang lebih verbose.

Contoh data mahasiswa dalam JSON:

```
{
  "nim": "2026001",
  "nama": "Alya Pratama",
  "prodi": "Informatika",
  "minat": ["database", "python", "cybersecurity"]
}
```

Data di atas mudah dibaca. Kita tahu ada `nim`, `nama`, `prodi`, dan `minat`. Field `minat` berbentuk array karena satu mahasiswa bisa memiliki lebih dari satu minat.

Intinya: JSON membuat data lebih fleksibel, tetapi tetap harus disiplin.

1.2 Evolusi Data: Dari Tabel Relasional ke Data Semi-Terstruktur

Database relasional menggunakan tabel. Tabel memiliki kolom dan baris. Model ini sangat kuat untuk data yang rapi dan stabil.

Contoh tabel mahasiswa:

nim	nama	prodi
2026001	Alya Pratama	Informatika
2026002	Bima Santoso	Sistem Informasi

Model tabel seperti ini sangat baik untuk data yang jelas, konsisten, dan sering dicari.

Namun, aplikasi modern sering memiliki data yang lebih fleksibel.

Contoh:

- Profil mahasiswa punya daftar skill yang bisa berbeda-beda.
- Produk punya spesifikasi yang tidak selalu sama.
- API mengirim response dengan struktur bertingkat.
- Log sistem memiliki field yang berubah sesuai jenis event.
- Workflow automation mengirim payload berbeda antar proses.

Data seperti ini disebut **semi-terstruktur**. Tidak sepenuhnya bebas seperti teks biasa, tetapi juga tidak selalu cocok dipaksa menjadi kolom tetap.

Di sinilah JSON berguna.

Contoh profil mahasiswa dengan data semi-terstruktur:

```
{
  "alamat": {
    "kota": "Tangerang Selatan",
    "provinsi": "Banten"
  },
  "skill": ["Python", "SQL", "Linux"],
  "sertifikasi": [
    {
      "nama": "Database Fundamental",
      "tahun": 2026
    }
  ]
}
```

Data ini tetap punya struktur, tetapi strukturnya lebih lentur dibanding tabel biasa.

1.3 JSON sebagai Bahasa Pertukaran Data Modern

JSON banyak dipakai dalam:

- **API** untuk mengirim dan menerima data.
- **Webhook** untuk mengirim notifikasi antar sistem.
- **Dashboard** untuk membaca data dari backend.
- **Aplikasi mobile** untuk komunikasi dengan server.
- **Log sistem** untuk menyimpan event.
- **Workflow automation** untuk memindahkan data antar node.
- **Script Python** untuk membaca, membersihkan, dan mengirim data.

MySQL mendukung tipe data JSON native. Dokumentasi resmi MySQL menyatakan bahwa tipe data JSON memberikan validasi otomatis terhadap dokumen JSON yang disimpan; dokumen tidak valid akan menghasilkan error. (MySQL Developer Zone)

Ini penting. Jika JSON disimpan sebagai teks biasa, database hanya melihatnya sebagai string panjang. Tetapi jika disimpan sebagai tipe `JSON`, database dapat memvalidasi dan memprosesnya dengan fungsi JSON.

Pesan kritis: jangan menyimpan JSON sebagai teks biasa jika database sudah menyediakan tipe JSON yang benar.

1.4 Mengapa JSON Lebih Mudah Dibaca Dibanding Format yang Terlalu Verbose?

Format data yang terlalu verbose membuat manusia cepat lelah membaca. Banyak tag, banyak penutup, dan banyak simbol yang mengganggu fokus.

JSON lebih ringkas.

Contoh JSON:

```
{
  "nama": "Alya",
  "status": "aktif"
}
```

Strukturnya langsung terlihat. Ada key `nama`, nilainya `Alya`. Ada key `status`, nilainya `aktif`.

Keunggulan JSON:

- **Ringkas**.
- **Mudah dibaca manusia**.
- **Mudah diparse program**.
- **Cocok untuk API**.
- **Cocok untuk nested data**.
- **Cocok untuk data semi-terstruktur**.

Namun, JSON juga punya risiko:

- Key bisa tidak konsisten.
- Tipe data bisa berubah.
- Struktur bisa terlalu dalam.
- Query bisa lambat jika tidak didesain.
- Valid secara sintaks belum tentu benar secara bisnis.

Contoh JSON valid tetapi buruk:

```
{
  "nama": "Alya",
  "umur": "dua puluh",
  "email": "bukan-email"
}
```

Secara bentuk, JSON ini valid. Tetapi secara isi, bermasalah. `umur` seharusnya angka. `email` seharusnya format alamat email yang benar.

JSON valid belum tentu data benar.

1.5 Posisi MySQL dalam Aplikasi API, Backend, Dashboard, dan Otomasi

MySQL tetap penting karena banyak aplikasi membutuhkan database yang stabil, cepat, dan dapat dikelola dengan SQL.

Dalam aplikasi modern, MySQL dapat berperan sebagai:

- **Database utama** untuk data relasional.
- **Penyimpan data semi-terstruktur** melalui kolom JSON.
- **Sumber data dashboard**.
- **Tujuan akhir pipeline ETL sederhana**.
- **Penyimpan log aplikasi ringan**.
- **Backend database untuk API sederhana**.
- **Sumber data untuk workflow automation**.

MySQL Community Edition adalah versi yang dapat diunduh bebas dan berada di bawah lisensi GPL, sehingga cocok untuk pembelajaran dan eksperimen open source. (MySQL)

MySQL juga menyediakan fungsi JSON seperti `JSON_EXTRACT`. Operator `->` di MySQL berfungsi sebagai alias dari `JSON_EXTRACT()` ketika digunakan untuk mengambil data dari dokumen JSON dengan JSON path. (MySQL Developer Zone)

Contoh sederhana:

```
SELECT JSON_EXTRACT(profile, '$.nama') AS nama
FROM mahasiswa_profile;
```

Query ini mengambil nilai `nama` dari kolom JSON bernama `profile`.

1.6 Peran Python sebagai Scripting Engine

Python berperan sebagai alat kerja praktis untuk mahasiswa. Dengan Python, mahasiswa dapat membaca file JSON, membersihkan data, memvalidasi struktur, lalu memasukkannya ke MySQL.

Python cocok untuk:

- Membaca file `.json`.
- Membaca file `.jsonl`.
- Mengubah dictionary Python menjadi JSON.
- Mengambil data dari API.
- Membersihkan data sebelum masuk database.
- Mengirim query ke MySQL.
- Mengekspor hasil query kembali menjadi JSON.
- Membuat laporan sederhana.

Untuk menjaga lingkungan kerja tetap rapi, Python sebaiknya menggunakan *virtual environment*. Panduan Python Packaging menjelaskan penggunaan `venv` dan `pip` untuk membuat lingkungan terisolasi serta memasang package di dalamnya. (Panduan Pengemasan Python)

Contoh alur kerja Python:

```
file JSON → Python → validasi → MySQL → query → export JSON
```

Python adalah lem antara file, API, validasi, database, dan laporan.

1.7 Peran n8n sebagai Workflow Automation Engine

n8n berguna untuk mempercepat integrasi. Mahasiswa tidak harus selalu membuat seluruh alur dari nol dengan kode. Beberapa proses dapat dibuat sebagai workflow.

n8n dapat dipakai untuk:

- Menerima data JSON dari webhook.
- Membersihkan field.
- Mengirim data ke MySQL.
- Mengambil data dari API.
- Menjalankan query.
- Mengirim respons JSON.
- Membuat otomasi sederhana.

Dokumentasi n8n menyatakan bahwa n8n dapat diinstal secara self-hosted dan instalasi dapat dilakukan menggunakan `npm` atau pendekatan container. (n8n Docs) Untuk latihan lokal, dokumentasi n8n juga menyebut `npm` sebagai cara cepat memulai di mesin lokal, dengan syarat Node.js tersedia. (n8n Docs)

n8n juga memiliki node MySQL yang mendukung eksekusi query SQL, insert, dan update row. (n8n Docs)

Dalam buku ini, n8n tidak menggantikan Python. Perannya berbeda:

- **Python** cocok untuk logic yang detail, validasi, transformasi kompleks, dan script yang bisa diuji.
- **n8n** cocok untuk otomasi alur, webhook, integrasi cepat, dan prototyping workflow.

Strategi yang sehat: gunakan Python untuk logic penting, gunakan n8n untuk orkestrasi.

1.8 Kapan Memakai Kolom Relasional, Kapan Memakai Kolom JSON?

Ini bagian penting. Banyak mahasiswa terjebak: karena MySQL bisa menyimpan JSON, semua data dimasukkan ke JSON. Itu salah.

Kolom relasional biasa cocok untuk data yang:

- Selalu ada.
- Sering dicari.
- Sering difilter.
- Sering diurutkan.
- Dipakai untuk join.
- Punya tipe data jelas.
- Menjadi identitas utama.

Contoh kolom relasional:

```
id
nim
nama
prodi
email
created_at
```

Kolom JSON cocok untuk data yang:

- Fleksibel.
- Opsional.
- Berbeda antar record.
- Berisi nested object.
- Berisi array.
- Sering datang dari API.
- Tidak selalu dipakai untuk filter utama.

Contoh kolom JSON:

```
profile_extra
metadata
preferences
api_payload
log_detail
spesifikasi
```

Contoh desain hybrid:

```
CREATE TABLE mahasiswa_profile (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL,
  nama VARCHAR(100) NOT NULL,
  prodi VARCHAR(100) NOT NULL,
  profile JSON,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Desain ini lebih sehat. Data penting seperti `nim`, `nama`, dan `prodi` tetap menjadi kolom biasa. Data fleksibel masuk ke `profile`.

Prinsip desain: data inti masuk kolom biasa; data fleksibel masuk JSON.

1.9 Risiko Awal: Fleksibel Bukan Berarti Bebas

JSON memberi fleksibilitas, tetapi fleksibilitas tanpa disiplin akan merusak database.

Risiko utama:

- **Data berantakan** karena key tidak konsisten.
- **Validasi lemah** karena semua payload diterima.
- **Query lambat** karena field penting tersembunyi di JSON.
- **Struktur terlalu dalam** sehingga sulit dibaca.
- **Duplikasi data** karena tidak ada desain yang jelas.
- **Sulit membuat laporan** karena format antar record berbeda.
- **Sulit audit** karena perubahan struktur tidak terdokumentasi.

Contoh buruk:

```
{
  "Nama": "Alya",
  "nama_mahasiswa": "Alya Pratama",
  "studentName": "Alya P.",
  "email": "alya@example.test",
  "EmailMahasiswa": "alya@example.test"
}
```

Masalahnya bukan JSON. Masalahnya adalah tidak ada standar.

Contoh lebih baik:

```
{
  "nama_lengkap": "Alya Pratama",
  "email": "alya@example.test",
  "minat": ["database", "python"],
  "alamat": {
    "kota": "Tangerang Selatan",
    "provinsi": "Banten"
  }
}
```

JSON harus fleksibel, bukan liar.

Hands-on Bab 1

Bagian praktik ini membuat database latihan, tabel dengan kolom JSON, insert satu dokumen JSON, lalu mengambil data menggunakan `JSON_EXTRACT`.

Praktik 1 — Instalasi Ringkas MySQL Community Server

Pada Linux berbasis Debian/Ubuntu, instalasi dasar dapat dilakukan melalui paket sistem. Dokumentasi MySQL menyebut instalasi Linux dapat dilakukan melalui package manager distribusi, meskipun versi dari repository distribusi bisa tertinggal dibanding rilis terbaru. (MySQL Developer Zone)

```
sudo apt update
sudo apt install mysql-server mysql-client
sudo systemctl enable mysql
sudo systemctl start mysql
sudo systemctl status mysql
```

Penjelasan singkat:

- `sudo`: menjalankan perintah dengan hak administratif.
- `apt update`: memperbarui daftar paket.
- `apt install`: memasang paket.
- `mysql-server`: layanan database MySQL.
- `mysql-client`: alat command line untuk terhubung ke MySQL.
- `systemctl enable`: membuat service aktif saat sistem menyala.
- `systemctl start`: menjalankan service sekarang.
- `systemctl status`: mengecek status service.

Masuk ke MySQL:

```
sudo mysql
```

Cek versi:

```
SELECT VERSION();
```

Praktik 2 — Membuat User dan Database Latihan

Masuk ke MySQL, lalu jalankan:

```
CREATE DATABASE json_lab;
CREATE USER 'json_user'@'localhost' IDENTIFIED BY
'GantiPasswordKuat';
GRANT ALL PRIVILEGES ON json_lab.* TO
'json_user'@'localhost';
FLUSH PRIVILEGES;
```

Penjelasan singkat:

- `CREATE DATABASE`: membuat database baru.
- `json_lab`: nama database latihan.
- `CREATE USER`: membuat user database.
- `IDENTIFIED BY`: memberi password.
- `GRANT ALL PRIVILEGES`: memberi hak akses ke database latihan.
- `FLUSH PRIVILEGES`: memuat ulang hak akses.

Gunakan database:

```
USE json_lab;
```

Praktik 3 — Membuat Tabel Pertama dengan Kolom JSON

```
CREATE TABLE mahasiswa_profile (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL,
  nama VARCHAR(100) NOT NULL,
  prodi VARCHAR(100) NOT NULL,
  profile JSON,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Penjelasan bagian penting:

- `id`: nomor unik setiap baris.
- `INT`: tipe angka bulat.
- `AUTO_INCREMENT`: nilai otomatis naik.
- `PRIMARY KEY`: identitas utama baris.
- `nim`: nomor induk mahasiswa.
- `VARCHAR(20)`: teks maksimal 20 karakter.
- `NOT NULL`: wajib diisi.
- `profile JSON`: kolom untuk menyimpan dokumen JSON.
- `created_at`: waktu data dibuat.
- `DEFAULT CURRENT_TIMESTAMP`: otomatis berisi waktu saat insert.

Praktik 4 — Insert Satu Dokumen JSON Sederhana

```
INSERT INTO mahasiswa_profile (nim, nama, prodi, profile)
VALUES (
  '2026001',
  'Alya Pratama',
  'Informatika',
  JSON_OBJECT(
    'alamat', JSON_OBJECT(
      'kota', 'Tangerang Selatan',
      'provinsi', 'Banten'
    ),
    'minat', JSON_ARRAY('database', 'python',
'cybersecurity'),
    'angkatan', 2026
  )
);
```

Penjelasan:

- `INSERT INTO`: memasukkan data baru.
- `mahasiswa_profile`: nama tabel.
- `(nim, nama, prodi, profile)`: daftar kolom yang akan diisi.
- `VALUES`: nilai yang dimasukkan.
- `JSON_OBJECT`: membuat object JSON.
- `JSON_ARRAY`: membuat array JSON.
- `'alamat'`: key JSON.
- `'kota'`: key di dalam object `alamat`.
- `'minat'`: key yang berisi array.
- `2026`: angka, bukan string.

Praktik 5 — Query Pertama Menggunakan JSON_EXTRACT

```
SELECT
  nim,
  nama,
  JSON_EXTRACT(profile, '$.alamat.kota') AS kota
FROM mahasiswa_profile;
```

Penjelasan:

- `SELECT`: mengambil data.
- `nim, nama`: kolom relasional biasa.
- `JSON_EXTRACT`: mengambil nilai dari dokumen JSON.
- `profile`: nama kolom JSON.
- `'$.alamat.kota'`: JSON path menuju field `kota`.
- `AS kota`: memberi nama kolom hasil.

Hasilnya kurang lebih:

nim	nama	kota
2026001	Alya Pratama	"Tangerang Selatan"

Nilai masih memakai tanda kutip karena hasilnya masih format JSON. Untuk mengambil nilai teks bersih, gunakan:

```
SELECT
  nim,
  nama,
  JSON_UNQUOTE(JSON_EXTRACT(profile, '$.alamat.kota')) AS
kota
FROM mahasiswa_profile;
```

Atau gunakan operator:

```
SELECT
  nim,
  nama,
  profile->>'$.alamat.kota' AS kota
FROM mahasiswa_profile;
```

Dokumentasi MySQL menjelaskan bahwa operator `->` adalah alias untuk `JSON_EXTRACT`, sedangkan operator `->>` mengembalikan nilai yang sudah di-*unquote*. (MySQL Developer Zone)

Praktik 6 — Instalasi Ringkas Python dan Pustaka Database

```
sudo apt install python3 python3-venv python3-pip
python3 -m venv venv
source venv/bin/activate
pip install mysql-connector-python
```

Penjelasan:

- `python3`: menjalankan Python versi 3.
- `python3-venv`: modul untuk membuat lingkungan Python terpisah.
- `python3-pip`: alat instalasi package Python.
- `python3 -m venv venv`: membuat folder virtual environment bernama `venv`.
- `source venv/bin/activate`: mengaktifkan virtual environment.
- `pip install mysql-connector-python`: memasang pustaka koneksi MySQL.

Script uji koneksi:

```
import mysql.connector

conn = mysql.connector.connect(
    host="localhost",
    user="json_user",
    password="GantiPasswordKuat",
    database="json_lab"
)

cursor = conn.cursor(dictionary=True)
cursor.execute("SELECT nim, nama, profile->>'$.alamat.kota' AS kota FROM
mahasiswa_profile")

for row in cursor.fetchall():
    print(row)

cursor.close()
conn.close()
```

Penjelasan singkat:

- `import mysql.connector`: memanggil pustaka koneksi MySQL.
- `connect()`: membuka koneksi ke database.
- `host`: alamat server database.
- `user`: user database.
- `password`: password user.
- `database`: database yang dipakai.
- `cursor(dictionary=True)`: hasil query dikembalikan sebagai dictionary.
- `execute()`: menjalankan SQL.
- `fetchall()`: mengambil semua hasil.
- `print(row)`: menampilkan hasil.
- `close()`: menutup cursor dan koneksi.

Praktik 7 — Instalasi Ringkas n8n secara Lokal

Untuk latihan lokal, gunakan jalur `npm`.

```
npm install n8n -g
n8n
```

Penjelasan:

- `npm`: package manager untuk ekosistem Node.js.
- `install`: memasang package.
- `n8n`: nama package.
- `-g`: instalasi global agar perintah `n8n` bisa dipanggil dari terminal.
- `n8n`: menjalankan aplikasi n8n.

Setelah berjalan, buka antarmuka n8n di browser lokal. Buat workflow sederhana:

- **Webhook** menerima JSON.
- **Set** merapikan field.
- **MySQL** menyimpan data ke tabel.
- **Respond to Webhook** mengirim respons JSON.

Contoh payload yang dikirim ke webhook:

```
{
  "nim": "2026002",
  "nama": "Bima Santoso",
  "prodi": "Sistem Informasi",
  "profile": {
    "alamat": {
      "kota": "Depok",
      "provinsi": "Jawa Barat"
    },
    "minat": ["data", "api", "automation"],
    "angkatan": 2026
  }
}
```

Pada tahap ini, mahasiswa belum perlu membuat workflow kompleks. Tujuannya hanya memahami bahwa **JSON dapat bergerak dari webhook ke database**.

Mini-Lab Bab 1 — Database Profil Mahasiswa Berbasis JSON

Tujuan

Membuat database sederhana yang menyimpan data mahasiswa dalam desain hybrid: kolom relasional untuk data inti, kolom JSON untuk data fleksibel.

Langkah

1. Buat database `json_lab`.
2. Buat user `json_user`.
3. Buat tabel `mahasiswa_profile`.
4. Insert minimal 3 data mahasiswa.
5. Simpan `alamat`, `minat`, dan `angkatan` ke kolom JSON.
6. Query nama mahasiswa dan kota asal.
7. Query mahasiswa yang memiliki minat `python`.
8. Tulis hasil observasi pendek.

Contoh query tambahan:

```
SELECT
  nim,
  nama,
  profile->>'$.alamat.kota' AS kota,
  profile->>'$.angkatan' AS angkatan
FROM mahasiswa_profile;
```

Query untuk melihat JSON lengkap:

```
SELECT JSON_PRETTY(profile) AS profile_rapi
FROM mahasiswa_profile;
```

Kesalahan Umum Mahasiswa

Beberapa kesalahan yang sering terjadi:

- Menulis JSON dengan koma berlebih.
- Lupa tanda kutip pada key.
- Mencampur nama key: `nama`, `Nama`, `student_name`.
- Menyimpan data penting hanya di JSON.
- Tidak membuat user database khusus.
- Menggunakan password lemah.
- Tidak membedakan data inti dan data tambahan.
- Menganggap JSON sebagai pengganti semua tabel.
- Tidak mengecek hasil query dengan teliti.

Contoh JSON salah:

```
{
  "nama": "Alya",
  "prodi": "Informatika",
}
```

Masalahnya: ada koma berlebih setelah `"Informatika"`.

Contoh JSON benar:

```
{
  "nama": "Alya",
  "prodi": "Informatika"
}
```

Checklist Akhir Bab

Mahasiswa dinyatakan memahami Bab 1 jika mampu:

- Menjelaskan fungsi JSON dalam aplikasi modern.
- Menjelaskan perbedaan data relasional dan semi-terstruktur.
- Menjelaskan posisi MySQL dalam sistem API dan otomasi.
- Menjelaskan peran Python sebagai alat scripting.
- Menjelaskan peran n8n sebagai alat workflow.
- Membuat database latihan.
- Membuat tabel dengan kolom JSON.
- Insert data JSON.
- Query isi JSON dengan `JSON_EXTRACT`.
- Menjelaskan kapan JSON berguna dan kapan berbahaya.

Ringkasan Bab

JSON penting karena aplikasi modern tidak hanya menyimpan data, tetapi juga **bertukar data**. MySQL tetap kuat sebagai database relasional, tetapi kini dapat menangani data semi-terstruktur melalui tipe JSON. Python membantu membaca, membersihkan, memvalidasi, dan mengirim data. n8n membantu mengotomasi alur antar sistem. Namun, fleksibilitas JSON harus dikendalikan. Data inti tetap sebaiknya berada di kolom relasional. Data fleksibel dapat masuk ke kolom JSON. **JSON yang baik membuat sistem adaptif; JSON yang liar membuat sistem rapuh.**

Bab 2 — Dasar JSON di MySQL: Struktur, Validasi, dan JSON Path

> **JSON harus fleksibel, bukan liar. Fleksibel berarti mudah dikembangkan; liar berarti sulit dicari, sulit divalidasi, dan sulit dipercaya.**

Bab 2 diposisikan sebagai fondasi teknis agar mahasiswa memahami bahwa JSON di MySQL bukan teks panjang biasa, tetapi tipe data khusus yang divalidasi, diproses, dan diakses melalui *path*.

Tujuan Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa mampu:

- Menjelaskan struktur dasar **JSON object**, **array**, **key-value**, dan *nested object*.
- Membedakan tipe data **JSON**, **TEXT**, **VARCHAR**, dan **BLOB**.
- Memahami bahwa MySQL dapat melakukan **validasi otomatis** terhadap kolom JSON.
- Menggunakan fungsi `JSON_VALID`, `JSON_TYPE`, dan `JSON_EXTRACT`.
- Menggunakan operator `->` dan `->>`.
- Menulis *JSON path* seperti `$.nama`, `$.alamat.kota`, dan `$.items[0]`.
- Memahami bahwa indeks array JSON dimulai dari **0**.
- Mendeteksi kesalahan umum seperti quote salah, koma berlebih, dan key tidak konsisten.
- Membuat mini-lab **profil mahasiswa berbasis JSON**.

2.1 JSON Bukan Sekadar Teks Panjang

Banyak pemula menganggap JSON hanyalah teks yang bentuknya mirip konfigurasi. Ini berbahaya. Jika JSON diperlakukan seperti teks biasa, database tidak bisa memahami struktur di dalamnya secara optimal.

MySQL memiliki tipe data `JSON`. Artinya, MySQL tidak hanya menyimpan JSON sebagai karakter panjang, tetapi juga memeriksa apakah formatnya valid. Dokumentasi MySQL menjelaskan bahwa tipe data JSON memberikan validasi otomatis; dokumen JSON yang tidak valid akan ditolak. (MySQL Developer Zone)

Buku Stokes juga menegaskan bahwa MySQL menyimpan data JSON dalam format internal teroptimasi untuk pencarian cepat berdasarkan key atau posisi array. MySQL terlebih dahulu memeriksa validitas dokumen, lalu menyimpannya dalam format internal agar sub-object atau nilai nested dapat dicari tanpa membaca seluruh teks dari awal.

Makna praktisnya:

- JSON di MySQL **bukan sekadar string**.
- JSON dapat divalidasi oleh database.
- JSON dapat diakses dengan fungsi khusus.
- JSON dapat di navigasi menggunakan *path*.
- JSON yang buruk tetap akan menyulitkan query dan laporan.

2.2 Struktur Dasar JSON

JSON memiliki beberapa bentuk utama:

- **Object**
- **Array**
- **Key-value**
- **String**
- **Number**
- **Boolean**
- **Null**
- **Nested object**
- **Array of object**

Contoh JSON sederhana:

```
{
  "nim": "2026001",
  "nama": "Alya Pratama",
  "aktif": true,
  "angkatan": 2026
}
```

Penjelasan:

- `{ }` menandai **object**.
- `"nim"` adalah **key**.
- `"2026001"` adalah **value**.
- `"aktif": true` memakai nilai boolean.
- `"angkatan": 2026` memakai nilai angka.

Contoh JSON dengan array:

```
{
  "nama": "Alya Pratama",
  "skill": ["Python", "SQL", "Linux"]
}
```

Penjelasan:

- `skill` berisi **array**.
- Array ditulis dengan `[ ]`.
- Nilai di dalam array dipisahkan koma.
- Item pertama berada pada indeks **0**, bukan 1.

Contoh JSON dengan *nested object*:

```
{
  "nama": "Alya Pratama",
  "alamat": {
    "kota": "Tangerang Selatan",
    "provinsi": "Banten"
  }
}
```

Penjelasan:

- `alamat` adalah object di dalam object.
- `kota` berada di dalam `alamat`.
- Untuk mengambil `kota`, kita perlu menulis path: `$.alamat.kota`.

2.3 Object, Array, dan Key-Value

Object adalah kumpulan pasangan key dan value.

```
{
  "nama": "Bima",
  "prodi": "Informatika"
}
```

Array adalah daftar nilai berurutan.

```
["Python", "SQL", "Linux"]
```

Key-value adalah pasangan nama field dan isi.

```
"nama": "Bima"
```

Nested object adalah object di dalam object.

```
{
  "mahasiswa": {
    "nama": "Bima",
    "prodi": "Informatika"
  }
}
```

Array of object adalah array yang berisi beberapa object.

```
{
  "sertifikasi": [
    {
      "nama": "Database Fundamental",
      "tahun": 2026
    },
    {
      "nama": "Python Basic",
      "tahun": 2026
    }
  ]
}
```

Struktur seperti ini sering muncul dalam response API, data transaksi, profil pengguna, log aplikasi, dan konfigurasi workflow.

2.4 Perbedaan JSON, TEXT, VARCHAR, dan BLOB

Mahasiswa harus paham bahwa pilihan tipe data memengaruhi cara database membaca dan menjaga data.

Tipe Data	Fungsi Utama	Cocok Untuk	Risiko
`VARCHAR`	Teks pendek	nama, email, kode	tidak cocok untuk dokumen panjang
`TEXT`	Teks panjang	catatan, deskripsi	tidak memahami struktur JSON
`BLOB`	Data biner	file, gambar, data mentah	sulit dibaca langsung
`JSON`	Dokumen JSON valid	payload API, profil fleksibel, metadata	bisa kacau jika struktur tidak disiplin

Gunakan `JSON` jika data memang berbentuk JSON.

Jangan menyimpan JSON ke `TEXT` hanya karena lebih mudah. Itu membuat database kehilangan kemampuan validasi dan fungsi JSON native.

Contoh desain kurang baik:

```
CREATE TABLE mahasiswa_buruk (
  id INT AUTO_INCREMENT PRIMARY KEY,
  profile TEXT
);
```

Contoh desain lebih baik:

```
CREATE TABLE mahasiswa_profile (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL,
  nama VARCHAR(100) NOT NULL,
  profile JSON
);
```

Catatan kritis: tipe `JSON` bukan alasan untuk membuang desain relasional. Data penting tetap sebaiknya menjadi kolom biasa.

2.5 Validasi Otomatis JSON di MySQL

MySQL akan menolak dokumen JSON yang tidak valid ketika dimasukkan ke kolom bertipe `JSON`. Ini melindungi database dari kerusakan format dasar.

Contoh JSON valid:

```
{
  "nama": "Alya",
  "prodi": "Informatika"
}
```

Contoh JSON tidak valid:

```
{
  "nama": "Alya",
  "prodi": "Informatika",
}
```

Masalahnya ada pada koma terakhir setelah `"Informatika"`. JSON tidak menerima koma berlebih di akhir object.

Contoh insert yang benar:

```
INSERT INTO mahasiswa_profile (nim, nama, profile)
VALUES (
  '2026001',
  'Alya Pratama',
  '{"prodi": "Informatika", "angkatan": 2026}'
);
```

Contoh insert yang akan gagal:

```
INSERT INTO mahasiswa_profile (nim, nama, profile)
VALUES (
  '2026002',
  'Bima Santoso',
  '{"prodi": "Informatika", "angkatan": 2026,}'
);
```

Pesan penting: MySQL dapat memeriksa validitas sintaks JSON, tetapi belum tentu memeriksa kebenaran bisnis. JSON bisa valid secara format, tetapi isinya tetap salah.

Contoh:

```
{
  "email": "bukan-email",
  "umur": "dua puluh"
}
```

JSON ini valid secara sintaks, tetapi buruk secara data. Validasi bisnis akan dibahas lebih kuat pada Bab 8 dengan **JSON Schema**.

2.6 Fungsi Dasar JSON di MySQL

Bab ini fokus pada tiga fungsi dasar:

- `JSON_VALID`
- `JSON_TYPE`
- `JSON_EXTRACT`

Dokumentasi MySQL menjelaskan bahwa `JSON_TYPE()` mengembalikan tipe nilai JSON, misalnya object, array, atau scalar. (MySQL Developer Zone)

2.6.1 JSON_VALID

`JSON_VALID()` digunakan untuk mengecek apakah sebuah teks adalah JSON valid.

```
SELECT JSON_VALID('{"nama": "Alya"}') AS valid_json;
```

Hasil:

```
1
```

Nilai `1` berarti valid.

Contoh tidak valid:

```
SELECT JSON_VALID('{"nama": "Alya",}') AS valid_json;
```

Hasil:

```
0
```

Nilai `0` berarti tidak valid.

2.6.2 JSON_TYPE

`JSON_TYPE()` digunakan untuk mengetahui tipe nilai JSON.

```
SELECT JSON_TYPE('{"nama": "Alya"}') AS tipe;
```

Hasil:

```
OBJECT
```

Contoh array:

```
SELECT JSON_TYPE('["Python", "SQL", "Linux"]') AS tipe;
```

Hasil:

```
ARRAY
```

Contoh string:

```
SELECT JSON_TYPE('"Alya"') AS tipe;
```

Hasil:

```
STRING
```

Fungsi ini penting untuk debugging. Saat query tidak sesuai harapan, cek dulu tipe datanya.

2.6.3 JSON_EXTRACT

`JSON_EXTRACT()` digunakan untuk mengambil nilai dari dokumen JSON berdasarkan *path*.

```
SELECT JSON_EXTRACT('{"nama": "Alya"}', '$.nama') AS nama;
```

Hasil:

```
"Alya"
```

Nilai masih berbentuk JSON, sehingga string tetap memakai tanda kutip.

Untuk mengambil nilai bersih, gunakan:

```
SELECT JSON_UNQUOTE(JSON_EXTRACT('{"nama": "Alya"}',
'$.nama')) AS nama;
```

Hasil:

Alya

2.7 Operator -> dan ->>

MySQL menyediakan operator ringkas untuk mengambil nilai JSON.

Dokumentasi MySQL menjelaskan:

- `->` setara dengan `JSON_EXTRACT()`.
- `->>` setara dengan `JSON_UNQUOTE(JSON_EXTRACT())`. (MySQL Developer Zone)

Contoh:

```
SELECT profile->'$.alamat.kota' AS kota
FROM mahasiswa_profile;
```

Hasil:

```
"Tangerang Selatan"
```

Contoh dengan `->>`:

```
SELECT profile->>'$.alamat.kota' AS kota
FROM mahasiswa_profile;
```

Hasil:

```
Tangerang Selatan
```

Gunakan `->>` jika ingin hasil teks bersih.

Perbandingan cepat:

Bentuk	Fungsi	Hasil
`profile->'$.nama'`	Ambil nilai JSON	`"Alya"`
`profile->>'$.nama'`	Ambil nilai teks bersih	`Alya`
`JSON_EXTRACT(profile, '$.nama')`	Ambil nilai JSON	`"Alya"`
`JSON_UNQUOTE(JSON_EXTRACT(...))`	Ambil nilai teks bersih	`Alya`

2.8 Memahami JSON Path

JSON path adalah alamat menuju nilai tertentu di dalam dokumen JSON.

Dalam MySQL, path dimulai dengan simbol `$`.

Buku Stokes menjelaskan bahwa `$` merepresentasikan dokumen saat ini, lalu path dilanjutkan dengan key atau array index untuk menavigasi object dan array.

Contoh JSON:

```
{
  "nama": "Alya Pratama",
  "alamat": {
    "kota": "Tangerang Selatan",
    "provinsi": "Banten"
  },
  "skill": ["Python", "SQL", "Linux"]
}
```

Contoh path:

Path	Arti
`$.nama`	Ambil field `nama`
`$.alamat`	Ambil object `alamat`
`$.alamat.kota`	Ambil field `kota` di dalam `alamat`
`$.skill`	Ambil seluruh array skill
`$.skill[0]`	Ambil item pertama array skill
`$.skill[1]`	Ambil item kedua array skill
`$.skill[2]`	Ambil item ketiga array skill

Dokumentasi MySQL menegaskan bahwa posisi array dimulai dari nol. (MySQL Developer Zone)

Jadi:

- `skill[0]` = item pertama
- `skill[1]` = item kedua
- `skill[2]` = item ketiga

Kesalahan umum mahasiswa adalah mengambil item pertama dengan `[1]`. Itu salah dalam JSON path.

Hands-on Bab 2

Bagian praktik ini membuat tabel `mahasiswa_profile`, memasukkan profil mahasiswa dalam JSON, mengambil field tertentu, memfilter berdasarkan isi JSON, mendeteksi JSON invalid, dan memperbaiki data rusak.

2.9 Membuat Database dan Tabel mahasiswa_profile

Masuk ke MySQL:

```
mysql -u json_user -p
```

Gunakan database latihan:

```
USE json_lab;
```

Hapus tabel lama jika ada:

```
DROP TABLE IF EXISTS mahasiswa_profile;
```

Buat tabel baru:

```
CREATE TABLE mahasiswa_profile (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL,
  nama VARCHAR(100) NOT NULL,
  profile JSON NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Penjelasan:

- `id`: identitas unik.
- `nim`: data inti, disimpan sebagai kolom relasional.
- `nama`: data inti, disimpan sebagai kolom relasional.
- `profile`: data fleksibel, disimpan sebagai JSON.
- `created_at`: waktu data dibuat.

Desain ini hybrid: data penting tetap rapi di kolom biasa, data fleksibel masuk ke JSON.

2.10 Insert Profil Mahasiswa dalam Format JSON

Masukkan data pertama:

```
INSERT INTO mahasiswa_profile (nim, nama, profile)
VALUES (
  '2026001',
  'Alya Pratama',
  JSON_OBJECT(
    'prodi', 'Informatika',
    'angkatan', 2026,
    'alamat', JSON_OBJECT(
      'kota', 'Tangerang Selatan',
      'provinsi', 'Banten'
    ),
    'minat', JSON_ARRAY('database', 'python',
'cybersecurity'),
    'skill', JSON_ARRAY('Python', 'SQL', 'Linux')
  )
);
```

Masukkan data kedua:

```
INSERT INTO mahasiswa_profile (nim, nama, profile)
VALUES (
  '2026002',
  'Bima Santoso',
  JSON_OBJECT(
    'prodi', 'Sistem Informasi',
    'angkatan', 2026,
    'alamat', JSON_OBJECT(
      'kota', 'Depok',
      'provinsi', 'Jawa Barat'
    ),
    'minat', JSON_ARRAY('data', 'api', 'automation'),
    'skill', JSON_ARRAY('Python', 'MySQL', 'n8n')
  )
);
```

Masukkan data ketiga:

```
INSERT INTO mahasiswa_profile (nim, nama, profile)
VALUES (
  '2026003',
  'Citra Lestari',
  JSON_OBJECT(
    'prodi', 'Teknik Informatika',
    'angkatan', 2025,
    'alamat', JSON_OBJECT(
      'kota', 'Bogor',
      'provinsi', 'Jawa Barat'
    ),
    'minat', JSON_ARRAY('backend', 'database'),
    'skill', JSON_ARRAY('SQL', 'Python')
  )
);
```

Cek data:

```
SELECT id, nim, nama, JSON_PRETTY(profile) AS profile_rapi
FROM mahasiswa_profile;
```

`JSON_PRETTY()` membantu menampilkan JSON agar lebih mudah dibaca.

2.11 Query Nama, Alamat, Minat, dan Skill

Ambil nama dan kota:

```
SELECT
  nim,
  nama,
  profile->>'$.alamat.kota' AS kota
FROM mahasiswa_profile;
```

Ambil prodi dan angkatan:

```
SELECT
  nim,
  nama,
  profile->>'$.prodi' AS prodi,
  profile->>'$.angkatan' AS angkatan
FROM mahasiswa_profile;
```

Ambil seluruh array skill:

```
SELECT
  nim,
  nama,
  profile->'$.skill' AS skill
FROM mahasiswa_profile;
```

Ambil skill pertama:

```
SELECT
  nim,
  nama,
  profile->>'$.skill[0]' AS skill_pertama
FROM mahasiswa_profile;
```

Ambil skill kedua:

```
SELECT
  nim,
  nama,
  profile->>'$.skill[1]' AS skill_kedua
FROM mahasiswa_profile;
```

Catatan: `$.skill[0]` berarti item pertama, bukan item ke-0 secara logika manusia. Ini penting karena indeks array dimulai dari nol.

2.12 Filter Mahasiswa Berdasarkan Isi JSON

Filter berdasarkan kota:

```
SELECT
  nim,
  nama,
  profile->>'$.alamat.kota' AS kota
FROM mahasiswa_profile
WHERE profile->>'$.alamat.kota' = 'Depok';
```

Filter berdasarkan prodi:

```
SELECT
  nim,
  nama,
  profile->>'$.prodi' AS prodi
FROM mahasiswa_profile
WHERE profile->>'$.prodi' = 'Informatika';
```

Filter berdasarkan angkatan:

```
SELECT
  nim,
  nama,
  profile->>'$.angkatan' AS angkatan
FROM mahasiswa_profile
WHERE CAST(profile->>'$.angkatan' AS UNSIGNED) = 2026;
```

Mengapa perlu `CAST`?

> Karena `->>` mengembalikan nilai sebagai teks. Jika ingin membandingkan sebagai angka, ubah dulu menjadi angka.

Filter mahasiswa yang memiliki minat `database`:

```
SELECT
  nim,
  nama,
  profile->'$.minat' AS minat
FROM mahasiswa_profile
WHERE JSON_CONTAINS(profile->'$.minat',
JSON_QUOTE('database'));
```

Penjelasan:

- `JSON_CONTAINS` mengecek apakah JSON mengandung nilai tertentu.
- `profile->'$.minat'` mengambil array minat.
- `JSON_QUOTE('database')` membuat string menjadi nilai JSON yang valid.

Dokumentasi MySQL menjelaskan bahwa `JSON_CONTAINS()` digunakan untuk mengecek apakah dokumen JSON mengandung nilai tertentu pada path tertentu. (MySQL Developer Zone)

2.13 Mendeteksi JSON Valid dan Invalid

Coba cek JSON valid:

```
SELECT JSON_VALID('{"nama": "Alya", "angkatan": 2026}') AS
hasil;
```

Hasil:

```
1
```

Coba cek JSON invalid:

```
SELECT JSON_VALID('{"nama": "Alya", "angkatan": 2026,}') AS
hasil;
```

Hasil:

```
0
```

Cek tipe JSON:

```
SELECT JSON_TYPE('{"nama": "Alya"}') AS tipe;
```

Hasil:

```
OBJECT
```

Cek tipe array:

```
SELECT JSON_TYPE('["Python", "SQL"]') AS tipe;
```

Hasil:

```
ARRAY
```

Cek tipe angka:

```
SELECT JSON_TYPE('2026') AS tipe;
```

Hasil:

```
INTEGER
```

2.14 Latihan Memperbaiki JSON Rusak

Kasus 1 — Koma Berlebih

JSON rusak:

```
{
  "nama": "Alya",
  "prodi": "Informatika",
}
```

Perbaikan:

```
{
  "nama": "Alya",
  "prodi": "Informatika"
}
```

Kasus 2 — Key Tidak Memakai Double Quote

JSON rusak:

```
{
  nama: "Bima",
  "prodi": "Sistem Informasi"
}
```

Perbaikan:

```
{
  "nama": "Bima",
  "prodi": "Sistem Informasi"
}
```

Kasus 3 — String Tidak Memakai Double Quote

JSON rusak:

```
{
  "nama": "Citra",
  "prodi": Teknik Informatika
}
```

Perbaikan:

```
{
  "nama": "Citra",
  "prodi": "Teknik Informatika"
}
```

Kasus 4 — Key Tidak Konsisten

JSON valid tetapi buruk:

```
{
  "nama": "Alya",
  "NamaMahasiswa": "Alya Pratama",
  "student_name": "Alya P."
}
```

Perbaikan konseptual:

```
{
  "nama_lengkap": "Alya Pratama"
}
```

Masalah key tidak konsisten lebih berbahaya daripada error sintaks. Error sintaks langsung terlihat. Key tidak konsisten sering baru terlihat saat laporan gagal dibuat.

2.15 Validasi JSON dengan Python

Python memiliki modul `json` bawaan. Dokumentasi Python menjelaskan bahwa modul `json` dapat dipakai dari command line melalui `python -m json` untuk memvalidasi dan merapikan JSON. (Python documentation)

Buat file `profile.json`:

```
{
  "nama": "Alya Pratama",
  "prodi": "Informatika",
  "skill": ["Python", "SQL", "Linux"]
}
```

Validasi dari terminal:

```
python3 -m json.tool profile.json
```

Jika JSON valid, output akan ditampilkan rapi.

Jika JSON rusak, Python akan menampilkan error.

Contoh file rusak `profile-rusak.json`:

```
{
  "nama": "Alya Pratama",
  "prodi": "Informatika",
}
```

Jalankan:

```
python3 -m json.tool profile-rusak.json
```

Output akan menunjukkan lokasi kesalahan.

Prinsip kerja: validasi lebih baik dilakukan sebelum data masuk database.

Mini-Lab Bab 2 — Profil Mahasiswa Berbasis JSON

Tujuan

Mahasiswa membuat tabel profil mahasiswa yang memadukan kolom relasional dan kolom JSON. Mahasiswa belajar insert, query, filter, validasi, dan debugging JSON.

Dataset Mini

Gunakan tiga mahasiswa:

```
[
  {
    "nim": "2026001",
    "nama": "Alya Pratama",
    "profile": {
      "prodi": "Informatika",
      "angkatan": 2026,
      "alamat": {
        "kota": "Tangerang Selatan",
        "provinsi": "Banten"
      },
      "minat": ["database", "python", "cybersecurity"],
      "skill": ["Python", "SQL", "Linux"]
    }
  },
  {
    "nim": "2026002",
    "nama": "Bima Santoso",
    "profile": {
      "prodi": "Sistem Informasi",
      "angkatan": 2026,
      "alamat": {
        "kota": "Depok",
        "provinsi": "Jawa Barat"
      },
      "minat": ["data", "api", "automation"],
      "skill": ["Python", "MySQL", "n8n"]
    }
  },
  {
    "nim": "2026003",
    "nama": "Citra Lestari",
    "profile": {
      "prodi": "Teknik Informatika",
```

```
      "angkatan": 2025,
      "alamat": {
        "kota": "Bogor",
        "provinsi": "Jawa Barat"
      },
      "minat": ["backend", "database"],
      "skill": ["SQL", "Python"]
    }
  }
]
```

Tugas Praktik

Kerjakan:

- Buat tabel `mahasiswa_profile`.
- Insert tiga data mahasiswa.
- Tampilkan semua data dengan `JSON_PRETTY`.
- Ambil `nama`, `prodi`, dan `kota`.
- Ambil skill pertama dari setiap mahasiswa.
- Filter mahasiswa dari kota `Depok`.
- Filter mahasiswa angkatan `2026`.
- Filter mahasiswa yang memiliki minat `database`.
- Cek JSON valid dan invalid dengan `JSON_VALID`.
- Perbaiki tiga JSON rusak.

Query Jawaban Utama

Tampilkan profil rapi:

```
SELECT
  nim,
  nama,
  JSON_PRETTY(profile) AS profile_rapi
FROM mahasiswa_profile;
```

Ambil prodi dan kota:

```
SELECT
  nim,
  nama,
  profile->>'$.prodi' AS prodi,
  profile->>'$.alamat.kota' AS kota
FROM mahasiswa_profile;
```

Ambil skill pertama:

```
SELECT
  nim,
  nama,
  profile->>'$.skill[0]' AS skill_pertama
FROM mahasiswa_profile;
```

Filter angkatan 2026:

```
SELECT
  nim,
  nama,
  profile->>'$.angkatan' AS angkatan
FROM mahasiswa_profile
WHERE CAST(profile->>'$.angkatan' AS UNSIGNED) = 2026;
```

Filter minat database:

```
SELECT
  nim,
  nama,
  profile->'$.minat' AS minat
FROM mahasiswa_profile
WHERE JSON_CONTAINS(profile->'$.minat',
JSON_QUOTE('database'));
```

Kesalahan Umum Mahasiswa

Beberapa kesalahan yang harus dihindari:

- Mengira JSON sama dengan teks biasa.
- Memakai `TEXT` untuk data yang seharusnya `JSON`.
- Lupa bahwa key JSON harus memakai double quote.
- Menulis koma berlebih di akhir object atau array.
- Mengambil array pertama dengan `[1]`, padahal seharusnya `[0]`.
- Menggunakan key tidak konsisten seperti `nama`, `Nama`, `studentName`.
- Menyimpan semua data ke JSON tanpa kolom relasional.
- Tidak melakukan validasi sebelum insert.
- Tidak memahami beda `->` dan `->>`.
- Membandingkan angka JSON sebagai teks tanpa `CAST`.

Checklist Akhir Bab

Mahasiswa dinyatakan memahami Bab 2 jika mampu:

- Menjelaskan struktur **object**, **array**, dan **key-value**.
- Menjelaskan perbedaan `JSON`, `TEXT`, `VARCHAR`, dan `BLOB`.
- Membuat tabel dengan kolom JSON.
- Insert dokumen JSON valid.
- Menggunakan `JSON_VALID`.
- Menggunakan `JSON_TYPE`.
- Menggunakan `JSON_EXTRACT`.
- Menggunakan operator `->`.
- Menggunakan operator `->>`.
- Mengambil field nested dengan `$.alamat.kota`.
- Mengambil item array dengan `$.skill[0]`.
- Memfilter data berdasarkan isi JSON.
- Mengenali dan memperbaiki JSON rusak.

Ringkasan Bab

Bab ini membangun fondasi teknis JSON di MySQL. Mahasiswa belajar bahwa JSON bukan teks panjang biasa, tetapi tipe data khusus yang dapat divalidasi dan diproses oleh database. Struktur JSON terdiri dari object, array, key-value, dan nested object. MySQL menyediakan fungsi seperti `JSON_VALID`, `JSON_TYPE`, dan `JSON_EXTRACT`, serta operator ringkas `->` dan `->>`. Mahasiswa juga belajar menggunakan *JSON path* untuk membaca nilai seperti `$.nama`, `$.alamat.kota`, dan `$.skill[0]`. Pelajaran paling penting: **JSON memberi fleksibilitas, tetapi tanpa standar akan berubah menjadi kekacauan data.**

Bab 3 — CRUD JSON dengan MySQL: Insert, Read, Update, Delete

> **CRUD JSON yang baik bukan sekadar bisa insert dan update; CRUD JSON yang baik menjaga data tetap rapi, bisa dicari, bisa diperbaiki, dan tidak merusak desain database.**

Bab 3 difokuskan pada operasi **Insert, Read, Update, Delete** terhadap data JSON, termasuk `JSON_SET`, `JSON_REMOVE`, desain tabel hybrid, transaksi sederhana, serta studi kasus katalog produk berbasis kolom relasional dan kolom JSON.

Tujuan Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa mampu:

- Memahami pola **CRUD** untuk data JSON.
- Membuat tabel produk dengan desain **hybrid**.
- Menyimpan dokumen JSON ke kolom `spesifikasi`.
- Membaca field tertentu dari JSON.
- Mengubah sebagian isi JSON dengan `JSON_SET`.
- Menghapus elemen JSON dengan `JSON_REMOVE`.
- Menambahkan atribut baru tanpa mengubah struktur tabel.
- Menggunakan transaksi sederhana untuk operasi penting.
- Menilai kapan JSON membuat sistem lebih fleksibel.
- Menilai kapan JSON justru merusak desain database.

3.1 Pola CRUD untuk Data JSON

CRUD adalah pola dasar kerja database:

- **Create**: memasukkan data baru.
- **Read**: membaca data.
- **Update**: mengubah data.
- **Delete**: menghapus data.

Dalam tabel relasional biasa, CRUD dilakukan pada kolom tetap. Dalam JSON, CRUD bisa dilakukan pada **dokumen penuh** atau pada **bagian kecil di dalam dokumen**.

Contoh dokumen JSON produk:

```
{
  "warna": "hitam",
  "ram_gb": 8,
  "storage_gb": 256,
  "garansi_bulan": 12,
  "fitur": ["wifi", "bluetooth"]
}
```

Operasi CRUD terhadap JSON dapat berarti:

- Insert seluruh dokumen ke kolom JSON.
- Read hanya `warna`.
- Update hanya `ram_gb`.
- Tambah field baru seperti `berat_gram`.
- Hapus field seperti `garansi_bulan`.
- Tambah item baru ke array `fitur`.

MySQL menyediakan fungsi modifikasi JSON seperti `JSON_SET`, `JSON_INSERT`, `JSON_REPLACE`, dan `JSON_REMOVE`. Dokumentasi resmi MySQL menjelaskan bahwa `JSON_SET()` mengganti nilai yang sudah ada dan menambahkan nilai yang belum ada, sedangkan `JSON_INSERT()` hanya memasukkan nilai baru tanpa mengganti nilai lama, dan `JSON_REPLACE()` hanya mengganti nilai yang sudah ada. (MySQL Developer Zone)

3.2 Mengapa CRUD JSON Berbeda dari CRUD Tabel Biasa?

Pada tabel biasa, setiap atribut produk biasanya menjadi kolom.

Contoh:

```
id
nama_produk
kategori
harga
warna
ram_gb
storage_gb
garansi_bulan
```

Masalah muncul ketika produk berbeda jenis.

Contoh:

- Produk laptop punya `ram_gb`, `storage_gb`, `ukuran_layar`.
- Produk router punya `jumlah_port`, `standar_wifi`, `frekuensi`.
- Produk sensor punya `tegangan`, `akurasi`, `jenis_output`.
- Produk kamera punya `resolusi`, `lensa`, `night_mode`.

Jika semua atribut dipaksa menjadi kolom, tabel akan melebar dan banyak kolom kosong. Jika semua atribut dimasukkan ke JSON, query penting bisa menjadi sulit.

Solusi sehat adalah **desain hybrid**.

Data inti disimpan sebagai kolom relasional:

- `id`
- `nama_produk`
- `kategori`
- `harga`

Data fleksibel disimpan sebagai JSON:

- `spesifikasi`

Prinsipnya sederhana: data yang sering dicari menjadi kolom biasa; data yang fleksibel masuk JSON.

3.3 Desain Tabel Hybrid: Relasional + JSON

Tabel hybrid menggabungkan disiplin database relasional dan fleksibilitas JSON.

Contoh desain:

```
CREATE TABLE produk (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nama_produk VARCHAR(100) NOT NULL,
  kategori VARCHAR(50) NOT NULL,
  harga DECIMAL(12,2) NOT NULL,
  spesifikasi JSON NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Penjelasan:

- `id`: identitas unik produk.
- `nama_produk`: nama utama produk.
- `kategori`: kategori yang sering difilter.
- `harga`: angka penting untuk laporan dan transaksi.
- `spesifikasi`: atribut fleksibel berbasis JSON.
- `created_at`: waktu data dibuat.

Mengapa `harga` tidak dimasukkan ke JSON?

Karena harga biasanya:

- sering difilter,
- sering diurutkan,
- sering dihitung,
- sering dipakai laporan,
- sering dipakai transaksi.

Jadi, harga lebih sehat sebagai kolom relasional.

Mengapa `warna`, `ram_gb`, atau `garansi_bulan` bisa masuk JSON?

Karena atribut itu bisa berbeda antar jenis produk. Tidak semua produk memiliki RAM, tidak semua produk punya ukuran layar, dan tidak semua produk punya spesifikasi teknis yang sama.

3.4 Insert Dokumen JSON

Ada dua cara umum memasukkan JSON:

- memakai string JSON valid,
- memakai fungsi `JSON_OBJECT` dan `JSON_ARRAY`.

Contoh memakai string JSON:

```
INSERT INTO produk (nama_produk, kategori, harga,
spesifikasi)
VALUES (
  'Laptop Edukasi 14',
  'laptop',
  6500000,
  '{
    "warna": "abu-abu",
    "ram_gb": 8,
    "storage_gb": 256,
    "ukuran_layar_inci": 14,
    "garansi_bulan": 12,
    "fitur": ["wifi", "bluetooth", "kamera"]
  }'
);
```

Contoh memakai `JSON_OBJECT`:

```
INSERT INTO produk (nama_produk, kategori, harga,
spesifikasi)
VALUES (
  'Router Lab Kampus',
  'jaringan',
  850000,
  JSON_OBJECT(
    'warna', 'putih',
    'jumlah_port', 4,
    'standar_wifi', '802.11ac',
    'garansi_bulan', 12,
    'fitur', JSON_ARRAY('wifi', 'ethernet', 'manajemen_web')
  )
);
```

`JSON_OBJECT` dan `JSON_ARRAY` membantu mengurangi kesalahan format, terutama saat nilai JSON dibuat dari query atau script.

Buku Stokes menekankan bahwa JSON memiliki grammar sederhana: object memakai `{}`, array memakai `[]`, key dan value dipisahkan dengan titik dua, dan item dipisahkan koma. Struktur sederhana ini membuat JSON mudah dibaca dan mudah diproses oleh program.

3.5 Read Field JSON Tertentu

Membaca JSON penuh sering tidak efisien untuk laporan. Biasanya kita hanya membutuhkan field tertentu.

Ambil seluruh spesifikasi:

```
SELECT
  id,
  nama_produk,
  JSON_PRETTY(spesifikasi) AS spesifikasi_rapi
FROM produk;
```

Ambil warna:

```
SELECT
  nama_produk,
  spesifikasi->>'$.warna' AS warna
FROM produk;
```

Ambil RAM:

```
SELECT
  nama_produk,
  spesifikasi->>'$.ram_gb' AS ram_gb
FROM produk
WHERE kategori = 'laptop';
```

Ambil item pertama dari array fitur:

```
SELECT
  nama_produk,
  spesifikasi->>'$.fitur[0]' AS fitur_pertama
FROM produk;
```

Operator `->` adalah bentuk ringkas dari `JSON_EXTRACT`, sedangkan `->>` mengembalikan nilai yang sudah tidak dibungkus sebagai JSON string. Dokumentasi MySQL menempatkan operator dan fungsi JSON ini sebagai bagian dari *JSON function reference*. (MySQL Developer Zone)

3.6 Update Sebagian Isi JSON dengan JSON_SET

`JSON_SET` adalah fungsi inti untuk mengubah dokumen JSON.

Pola umum:

```
UPDATE nama_tabel
SET kolom_json = JSON_SET(kolom_json, '$.path', nilai_baru)
WHERE kondisi;
```

Contoh mengubah warna:

```
UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi, '$.warna', 'hitam')
WHERE nama_produk = 'Laptop Edukasi 14';
```

Contoh mengubah RAM:

```
UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi, '$.ram_gb', 16)
WHERE nama_produk = 'Laptop Edukasi 14';
```

Contoh mengubah garansi:

```
UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi, '$.garansi_bulan',
24)
WHERE nama_produk = 'Laptop Edukasi 14';
```

Contoh menambahkan field baru:

```
UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi, '$.berat_gram', 1350)
WHERE nama_produk = 'Laptop Edukasi 14';
```

Kekuatan `JSON_SET`: jika path sudah ada, nilainya diganti; jika path belum ada, field baru ditambahkan. Stokes menjelaskan bahwa `JSON_SET()` dapat mengganti nilai pada path yang sudah ada dan menambahkan key baru jika path belum ada.

Makna praktis: struktur tabel tidak perlu diubah hanya untuk menambahkan atribut baru di JSON.

3.7 Update Nested Object

JSON sering memiliki object di dalam object.

Contoh struktur:

```
{
  "dimensi": {
    "panjang_cm": 32,
    "lebar_cm": 21,
    "tinggi_cm": 2
  }
}
```

Tambahkan object `dimensi`:

```
UPDATE produk
SET spesifikasi = JSON_SET(
  spesifikasi,
  '$.dimensi',
  JSON_OBJECT(
    'panjang_cm', 32,
    'lebar_cm', 21,
    'tinggi_cm', 2
  )
)
WHERE nama_produk = 'Laptop Edukasi 14';
```

Update hanya tinggi:

```
UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi,
'$.dimensi.tinggi_cm', 1.8)
WHERE nama_produk = 'Laptop Edukasi 14';
```

Ini lebih aman daripada mengambil JSON penuh ke aplikasi, mengubahnya di luar database, lalu menulis ulang seluruh dokumen tanpa kontrol.

3.8 Menambahkan Array dan Object Baru

Tambahkan array `tag`:

```
UPDATE produk
SET spesifikasi = JSON_SET(
  spesifikasi,
  '$.tag',
  JSON_ARRAY('edukasi', 'lab', 'portable')
)
WHERE nama_produk = 'Laptop Edukasi 14';
```

Tambahkan object `supplier`:

```
UPDATE produk
SET spesifikasi = JSON_SET(
  spesifikasi,
  '$.supplier',
  JSON_OBJECT(
    'nama', 'CV Sumber Perangkat',
    'kota', 'Tangerang Selatan'
  )
)
WHERE nama_produk = 'Laptop Edukasi 14';
```

Tambahkan item baru ke array dapat dilakukan dengan fungsi khusus seperti `JSON_ARRAY_APPEND`. Dokumentasi MySQL memasukkan `JSON_ARRAY_APPEND()` sebagai salah satu fungsi untuk memodifikasi nilai JSON. (MySQL Developer Zone)

Contoh:

```
UPDATE produk
SET spesifikasi = JSON_ARRAY_APPEND(spesifikasi, '$.fitur',
'usb-c')
WHERE nama_produk = 'Laptop Edukasi 14';
```

Cek hasil:

```
SELECT
  nama_produk,
  JSON_PRETTY(spesifikasi) AS spesifikasi_rapi
FROM produk
WHERE nama_produk = 'Laptop Edukasi 14';
```

3.9 Menghapus Elemen JSON dengan JSON_REMOVE

`JSON_REMOVE` digunakan untuk menghapus field atau elemen tertentu dari dokumen JSON.

Pola umum:

```
UPDATE nama_tabel
SET kolom_json = JSON_REMOVE(kolom_json, '$.path')
WHERE kondisi;
```

Contoh menghapus garansi:

```
UPDATE produk
SET spesifikasi = JSON_REMOVE(spesifikasi, '$.garansi_bulan')
WHERE nama_produk = 'Laptop Edukasi 14';
```

Contoh menghapus object supplier:

```
UPDATE produk
SET spesifikasi = JSON_REMOVE(spesifikasi, '$.supplier')
WHERE nama_produk = 'Laptop Edukasi 14';
```

Contoh menghapus item pertama dalam array fitur:

```
UPDATE produk
SET spesifikasi = JSON_REMOVE(spesifikasi, '$.fitur[0]')
WHERE nama_produk = 'Laptop Edukasi 14';
```

Stokes menjelaskan bahwa `JSON_REMOVE()` digunakan untuk menghapus data dari dokumen JSON; jika elemen yang dituju tidak ada, operasi tersebut tidak dianggap error dan dokumen tidak berubah.

Catatan kritis: menghapus item array bisa mengubah posisi index. Setelah `fitur[0]` dihapus, item berikutnya dapat bergeser. Jangan mengandalkan posisi array untuk data yang sangat penting.

3.10 Delete Row vs Delete Item JSON

Ada dua jenis penghapusan:

- Menghapus **bagian JSON**.
- Menghapus **baris produk**.

Menghapus bagian JSON:

```
UPDATE produk
SET spesifikasi = JSON_REMOVE(spesifikasi, '$.warna')
WHERE id = 1;
```

Menghapus baris produk:

```
DELETE FROM produk
WHERE id = 1;
```

Gunakan `JSON_REMOVE` jika produk masih ada, tetapi atribut tertentu ingin dihapus.
Gunakan `DELETE` jika seluruh produk memang harus dihapus dari tabel.

Prinsip aman: jangan memakai `DELETE` jika yang ingin dihapus hanya field kecil di dalam JSON.

3.11 Delete Item Berdasarkan Kondisi Isi JSON

Contoh: hapus produk yang RAM-nya 4 GB.

```
DELETE FROM produk
WHERE kategori = 'laptop'
  AND CAST(spesifikasi->>'$.ram_gb' AS UNSIGNED) = 4;
```

Contoh: hapus produk yang tidak memiliki garansi.

```
DELETE FROM produk
WHERE spesifikasi->'$.garansi_bulan' IS NULL;
```

Namun, operasi `DELETE` harus hati-hati. Biasakan cek dulu dengan `SELECT`.

```
SELECT
  id,
  nama_produk,
  spesifikasi->>'$.ram_gb' AS ram_gb
FROM produk
WHERE kategori = 'laptop'
  AND CAST(spesifikasi->>'$.ram_gb' AS UNSIGNED) = 4;
```

Setelah yakin, baru lakukan `DELETE`.

Aturan aman: `SELECT` dulu, `DELETE` kemudian.

3.12 Transaction Sederhana untuk Operasi Penting

Transaksi digunakan agar beberapa operasi penting dapat dianggap sebagai satu kesatuan. Jika semua benar, jalankan `COMMIT`. Jika ada kesalahan, jalankan `ROLLBACK`.

Dokumentasi MySQL membahas perintah `START TRANSACTION`, `COMMIT`, dan `ROLLBACK` sebagai bagian dari *transactional statements*. (MySQL Developer Zone)

Contoh transaksi:

```
START TRANSACTION;

UPDATE produk
SET harga = 7000000
WHERE nama_produk = 'Laptop Edukasi 14';

UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi, '$.garansi_bulan',
24)
WHERE nama_produk = 'Laptop Edukasi 14';

COMMIT;
```

Jika ada kesalahan sebelum `COMMIT`, batalkan:

```
ROLLBACK;
```

Contoh latihan aman:

```
START TRANSACTION;

UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi, '$.status_uji',
'sementara')
WHERE nama_produk = 'Laptop Edukasi 14';

SELECT
  nama_produk,
  spesifikasi->>'$.status_uji' AS status_uji
FROM produk
WHERE nama_produk = 'Laptop Edukasi 14';

ROLLBACK;
```

Setelah `ROLLBACK`, perubahan dibatalkan.

Pesan penting: transaksi adalah rem darurat saat data penting sedang diubah.

3.13 Kapan JSON Membuat Sistem Lebih Fleksibel?

JSON berguna ketika struktur data memang berubah-ubah.

JSON cocok untuk:

- Spesifikasi produk yang berbeda antar kategori.
- Metadata tambahan dari API.
- Preferensi pengguna.
- Log aplikasi.
- Payload webhook.
- Data eksperimen.
- Data formulir yang masih sering berubah.
- Prototipe sistem yang belum stabil.

Contoh bagus:

```
{
  "warna": "hitam",
  "ram_gb": 16,
  "storage_gb": 512,
  "fitur": ["wifi", "bluetooth", "usb-c"]
}
```

Kenapa bagus?

- Key jelas.
- Tipe data masuk akal.
- Tidak terlalu dalam.
- Field fleksibel tetapi tetap rapi.

3.14 Kapan JSON Merusak Desain Database?

JSON menjadi berbahaya jika dipakai untuk menyembunyikan semua data.

Contoh buruk:

```
CREATE TABLE produk_buruk (
  id INT AUTO_INCREMENT PRIMARY KEY,
  data JSON NOT NULL
);
```

Masalah:

- `nama_produk` tersembunyi.
- `kategori` tersembunyi.
- `harga` tersembunyi.
- filter lebih sulit,
- laporan lebih rumit,
- index lebih sulit,
- validasi bisnis lebih lemah.

Buku Stokes mengingatkan bahwa menyimpan JSON dalam string lama membuat data sulit divalidasi, sulit menjaga konsistensi tag, dan menyakitkan untuk dicari; masalah serupa tetap bisa muncul jika JSON native dipakai tanpa desain yang disiplin.

JSON merusak desain jika:

- semua data dimasukkan ke satu kolom JSON,
- field penting tidak menjadi kolom relasional,
- key tidak konsisten,
- tipe data tidak konsisten,
- struktur terlalu dalam,
- query laporan menjadi rumit,
- tidak ada validasi,
- tidak ada strategi index.

Kesimpulan strategis: JSON adalah alat, bukan pelarian dari desain database.

Hands-on Bab 3

3.15 Membuat Tabel Produk

Masuk ke MySQL:

```
mysql -u json_user -p
```

Gunakan database latihan:

```
USE json_lab;
```

Hapus tabel lama jika ada:

```
DROP TABLE IF EXISTS produk;
```

Buat tabel baru:

```
CREATE TABLE produk (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nama_produk VARCHAR(100) NOT NULL,
  kategori VARCHAR(50) NOT NULL,
  harga DECIMAL(12,2) NOT NULL,
  spesifikasi JSON NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3.16 Insert Produk Elektronik dengan Spesifikasi Fleksibel

Produk 1:

```
INSERT INTO produk (nama_produk, kategori, harga,
spesifikasi)
VALUES (
  'Laptop Edukasi 14',
  'laptop',
  6500000,
  JSON_OBJECT(
    'warna', 'abu-abu',
    'ram_gb', 8,
    'storage_gb', 256,
    'ukuran_layar_inci', 14,
    'garansi_bulan', 12,
    'fitur', JSON_ARRAY('wifi', 'bluetooth', 'kamera')
  )
);
```

Produk 2:

```
INSERT INTO produk (nama_produk, kategori, harga,
spesifikasi)
VALUES (
  'Router Lab Kampus',
  'jaringan',
  850000,
  JSON_OBJECT(
    'warna', 'putih',
    'jumlah_port', 4,
    'standar_wifi', '802.11ac',
    'garansi_bulan', 12,
    'fitur', JSON_ARRAY('wifi', 'ethernet', 'manajemen_web')
  )
);
```

Produk 3:

```
INSERT INTO produk (nama_produk, kategori, harga,
spesifikasi)
VALUES (
  'Sensor Suhu IoT',
  'sensor',
  125000,
  JSON_OBJECT(
    'warna', 'biru',
    'tegangan_volt', 3.3,
    'akurasi_derajat', 0.5,
    'jenis_output', 'digital',
    'fitur', JSON_ARRAY('low_power', 'iot_ready')
  )
);
```

Cek hasil:

```
SELECT
  id,
  nama_produk,
  kategori,
  harga,
  JSON_PRETTY(spesifikasi) AS spesifikasi
FROM produk;
```

3.17 Read Field JSON Produk

Ambil warna semua produk:

```
SELECT
  nama_produk,
  kategori,
  spesifikasi->>'$.warna' AS warna
FROM produk;
```

Ambil RAM khusus laptop:

```
SELECT
  nama_produk,
  spesifikasi->>'$.ram_gb' AS ram_gb
FROM produk
WHERE kategori = 'laptop';
```

Ambil jumlah port router:

```
SELECT
  nama_produk,
  spesifikasi->>'$.jumlah_port' AS jumlah_port
FROM produk
WHERE kategori = 'jaringan';
```

Ambil fitur pertama:

```
SELECT
  nama_produk,
  spesifikasi->>'$.fitur[0]' AS fitur_pertama
FROM produk;
```

3.18 Update Stok, Warna, RAM, Ukuran Layar, dan Garansi

Tambahkan stok ke semua produk:

```
UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi, '$.stok', 10);
```

Update warna laptop:

```
UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi, '$.warna', 'hitam')
WHERE nama_produk = 'Laptop Edukasi 14';
```

Update RAM laptop:

UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi, '$.ram_gb', 16)
WHERE nama_produk = 'Laptop Edukasi 14';

Update ukuran layar:

```
UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi,
'$.ukuran_layar_inci', 15.6)
WHERE nama_produk = 'Laptop Edukasi 14';
```

Update garansi:

```
UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi, '$.garansi_bulan',
24)
WHERE nama_produk = 'Laptop Edukasi 14';
```

Cek hasil:

```
SELECT
  nama_produk,
  spesifikasi->>'$.warna' AS warna,
  spesifikasi->>'$.ram_gb' AS ram_gb,
  spesifikasi->>'$.ukuran_layar_inci' AS layar,
  spesifikasi->>'$.garansi_bulan' AS garansi
FROM produk
WHERE nama_produk = 'Laptop Edukasi 14';
```

3.19 Menambahkan Atribut Baru Tanpa Mengubah Struktur Tabel

Tambahkan `berat_gram`:

```
UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi, '$.berat_gram', 1350)
WHERE nama_produk = 'Laptop Edukasi 14';
```

Tambahkan `sertifikasi`:

```
UPDATE produk
SET spesifikasi = JSON_SET(
  spesifikasi,
  '$.sertifikasi',
  JSON_ARRAY('hemat_energi', 'layak_lab')
)
WHERE nama_produk = 'Laptop Edukasi 14';
```

Tambahkan object `pengujian`:

```
UPDATE produk
SET spesifikasi = JSON_SET(
  spesifikasi,
  '$.pengujian',
  JSON_OBJECT(
    'status', 'lulus',
    'tanggal', '2026-05-26'
  )
)
WHERE nama_produk = 'Laptop Edukasi 14';
```

Cek dokumen:

```
SELECT JSON_PRETTY(spesifikasi) AS spesifikasi
FROM produk
WHERE nama_produk = 'Laptop Edukasi 14';
```

3.20 Menghapus Elemen JSON

Hapus field `warna` dari sensor:

```
UPDATE produk
SET spesifikasi = JSON_REMOVE(spesifikasi, '$.warna')
WHERE nama_produk = 'Sensor Suhu IoT';
```

Hapus field `pengujian` dari laptop:

```
UPDATE produk
SET spesifikasi = JSON_REMOVE(spesifikasi, '$.pengujian')
WHERE nama_produk = 'Laptop Edukasi 14';
```

Hapus fitur pertama dari router:

```
UPDATE produk
SET spesifikasi = JSON_REMOVE(spesifikasi, '$.fitur[0]')
WHERE nama_produk = 'Router Lab Kampus';
```

Cek hasil:

```
SELECT
  nama_produk,
  JSON_PRETTY(spesifikasi) AS spesifikasi
FROM produk;
```

3.21 Delete Item Berdasarkan Kondisi Isi JSON

Cek produk dengan garansi 12 bulan:

```
SELECT
  id,
  nama_produk,
  spesifikasi->>'$.garansi_bulan' AS garansi
FROM produk
WHERE CAST(spesifikasi->>'$.garansi_bulan' AS UNSIGNED) = 12;
```

Hapus produk dengan harga murah dan tanpa garansi:

```
DELETE FROM produk
WHERE harga < 200000
  AND spesifikasi->'$.garansi_bulan' IS NULL;
```

Catatan: query `DELETE` ini hanya contoh. Dalam praktik nyata, lebih baik memakai status seperti `arsip` daripada langsung menghapus data penting.

3.22 Praktik Transaction Aman

Mulai transaksi:

```
START TRANSACTION;
```

Update harga dan garansi:

```
UPDATE produk
SET harga = 7000000
WHERE nama_produk = 'Laptop Edukasi 14';

UPDATE produk
SET spesifikasi = JSON_SET(spesifikasi, '$.garansi_bulan',
36)
WHERE nama_produk = 'Laptop Edukasi 14';
```

Cek hasil sementara:

```
SELECT
  nama_produk,
  harga,
  spesifikasi->>'$.garansi_bulan' AS garansi
FROM produk
WHERE nama_produk = 'Laptop Edukasi 14';
```

Simpan perubahan:

```
COMMIT;
```

Atau batalkan:

```
ROLLBACK;
```

Latihan penting: ulangi transaksi di atas, tetapi gunakan `ROLLBACK`, lalu cek apakah data kembali seperti sebelum transaksi.

Mini-Lab Bab 3 — Katalog Produk Sederhana Berbasis MySQL JSON

Tujuan

Mahasiswa membuat katalog produk sederhana dengan desain hybrid. Data inti disimpan di kolom biasa. Spesifikasi fleksibel disimpan dalam JSON.

Tugas

Kerjakan langkah berikut:

- Buat tabel `produk`.
- Insert minimal 5 produk dari kategori berbeda.
- Setiap produk wajib memiliki `spesifikasi` JSON.
- Ambil daftar produk dan warna.
- Ambil field spesifik berdasarkan kategori.
- Update minimal 3 field JSON memakai `JSON_SET`.
- Tambahkan field baru tanpa `ALTER TABLE`.
- Tambahkan array `tag`.
- Hapus satu field JSON memakai `JSON_REMOVE`.
- Gunakan transaksi untuk update harga dan garansi.
- Buat laporan singkat: kapan JSON membantu, kapan JSON menyulitkan.

Contoh Produk Tambahan

Produk kamera:

```
INSERT INTO produk (nama_produk, kategori, harga,
spesifikasi)
VALUES (
  'Kamera Lab Visual',
  'kamera',
  1200000,
  JSON_OBJECT(
    'resolusi_mp', 12,
    'night_mode', true,
    'koneksi', JSON_ARRAY('usb', 'wifi'),
    'garansi_bulan', 12
  )
);
```

Produk alat ukur:

```
INSERT INTO produk (nama_produk, kategori, harga,
spesifikasi)
VALUES (
  'Multimeter Digital',
  'alat_ukur',
  250000,
  JSON_OBJECT(
    'akurasi_persen', 0.5,
    'baterai', '9V',
    'fitur', JSON_ARRAY('ukur_tegangan', 'ukur_arus',
'ukur_resistansi')
  )
);
```

Query Laporan Mini

Daftar produk dan harga:

```
SELECT
  id,
  nama_produk,
  kategori,
  harga
FROM produk
ORDER BY kategori, harga;
```

Daftar produk dengan garansi:

```
SELECT
  nama_produk,
  kategori,
  spesifikasi->>'$.garansi_bulan' AS garansi_bulan
FROM produk
WHERE spesifikasi->'$.garansi_bulan' IS NOT NULL;
```

Daftar produk yang memiliki fitur `wifi`:

```
SELECT
  nama_produk,
  kategori,
  spesifikasi->'$.fitur' AS fitur
FROM produk
WHERE JSON_CONTAINS(spesifikasi->'$.fitur',
JSON_QUOTE('wifi'));
```

Daftar produk dengan spesifikasi rapi:

```
SELECT
  nama_produk,
  JSON_PRETTY(spesifikasi) AS spesifikasi
FROM produk;
```

Kesalahan Umum Mahasiswa

Hindari kesalahan berikut:

- Menyimpan semua data produk ke JSON.
- Menyimpan harga hanya di JSON.
- Membuat key tidak konsisten: `ram`, `RAM`, `ram_gb`, `memory`.
- Mengubah seluruh JSON hanya untuk update satu field kecil.
- Menghapus baris produk padahal hanya ingin menghapus atribut JSON.
- Tidak mengecek dengan `SELECT` sebelum `DELETE`.
- Tidak memakai transaksi untuk update penting.
- Menaruh data yang sering dicari di JSON tanpa strategi index.
- Membuat JSON terlalu dalam dan sulit dibaca.
- Menganggap fleksibilitas sebagai alasan untuk tidak mendesain.

Checklist Akhir Bab

Mahasiswa dinyatakan memahami Bab 3 jika mampu:

- Menjelaskan pola **CRUD JSON**.
- Membuat tabel hybrid `produk`.
- Insert data produk dengan `JSON_OBJECT`.
- Membaca field JSON dengan `->` dan `->>`.
- Mengubah field JSON dengan `JSON_SET`.
- Menambahkan object dan array baru.
- Menghapus field JSON dengan `JSON_REMOVE`.
- Membedakan `JSON_REMOVE` dan `DELETE`.
- Menggunakan `START TRANSACTION`, `COMMIT`, dan `ROLLBACK`.
- Menjelaskan kapan JSON membantu desain.
- Menjelaskan kapan JSON merusak desain.

Ringkasan Bab

Bab ini mengubah pemahaman JSON menjadi keterampilan nyata. Mahasiswa belajar membuat tabel produk dengan desain **hybrid**, menyimpan atribut fleksibel dalam kolom JSON, membaca field tertentu, mengubah sebagian isi dokumen dengan `JSON_SET`, dan menghapus elemen dengan `JSON_REMOVE`. JSON membuat database lebih adaptif ketika data memang bervariasi, tetapi dapat merusak desain jika semua data dipaksa masuk ke satu kolom JSON. **Kunci Bab 3 adalah disiplin: data inti tetap relasional, data fleksibel masuk JSON, dan setiap perubahan penting harus bisa dilacak, diuji, dan dibatalkan bila salah.**

Bab 4 — Python untuk Interaksi MySQL JSON

> **Python adalah jembatan kerja: JSON dibaca sebagai data, divalidasi sebagai struktur, disimpan ke MySQL secara aman, lalu diekspor kembali sebagai informasi yang bisa dipakai.**

Bab 4 menjadi **jantung praktek pemrograman**, dengan fokus koneksi Python ke MySQL, `mysql-connector-python`, *cursor*, *prepared statement*, import/export JSON, *error handling*, transaksi, *rollback*, dan *batch insert*.

Tujuan Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa mampu:

- Membuat koneksi Python ke MySQL.
- Menggunakan pustaka `mysql-connector-python`.
- Menulis struktur dasar script database.
- Membedakan **cursor biasa** dan **cursor dictionary**.
- Menggunakan **prepared statement** untuk mengurangi risiko *SQL injection*.
- Mengubah Python `dict` dan `list` menjadi JSON.
- Membaca file `students.json`.
- Insert banyak data JSON ke tabel MySQL.
- Query field JSON dari Python.
- Export hasil query menjadi `result.json`.
- Menangani error koneksi, error SQL, dan JSON invalid.
- Menggunakan transaksi dan `rollback`.
- Membuat mini-lab **importer JSON mahasiswa**.

4.1 Mengapa Python Penting untuk MySQL JSON?

MySQL kuat untuk menyimpan dan mengambil data. JSON kuat untuk pertukaran data. Python kuat untuk menghubungkan keduanya.

Dalam proyek nyata, data sering datang dari:

- file `.json`;
- file `.jsonl`;
- API;
- *webhook*;
- form aplikasi;
- log sistem;
- workflow otomasi;
- hasil scraping legal;
- hasil export sistem lain.

Python berperan sebagai **scripting engine**:

- membaca data;
- membersihkan data;
- mengecek struktur;
- mengubah format;
- menyimpan ke database;
- mengambil hasil query;
- menulis ulang ke JSON;
- membuat laporan sederhana.

Modul `json` di Python dapat melakukan encoding dan decoding JSON, serta mempertahankan urutan input dan output selama struktur datanya mendukung. Dokumentasi Python juga menyediakan `json.load`, `json.loads`, `json.dump`, dan `json.dumps` sebagai fungsi utama untuk membaca dan menulis JSON. (Python documentation)

Pesan utama: Python membuat JSON dan MySQL bisa bekerja sebagai satu pipeline.

4.2 Alur Kerja Python + JSON + MySQL

Alur umum bab ini:

```
students.json
    ↓
Python membaca file
    ↓
Python validasi struktur dasar
    ↓
Python insert ke MySQL
    ↓
MySQL menyimpan kolom relasional + kolom JSON
    ↓
Python query field JSON
    ↓
Python export result.json
```

Alur ini sederhana, tetapi sangat penting. Inilah pola dasar banyak aplikasi data modern.

Bab 4 memang diarahkan untuk membaca file JSON, mengirimnya ke MySQL, mengambil data, lalu mengekspor kembali ke JSON.

4.3 Instalasi mysql-connector-python

Gunakan *virtual environment* agar lingkungan kerja rapi.

python3 -m venv venv
source venv/bin/activate
pip install mysql-connector-python

Penjelasan singkat:

- `python3 -m venv venv`: membuat lingkungan Python terpisah.
- `source venv/bin/activate`: mengaktifkan lingkungan tersebut.
- `pip install mysql-connector-python`: memasang pustaka koneksi MySQL.

Cek instalasi:

```
python -c "import mysql.connector; print('connector siap')"
```

Jika muncul:

```
connector siap
```

berarti pustaka sudah bisa dipakai.

4.4 Struktur Dasar Script Database

Buat file `db_test.py`.

```
import mysql.connector

conn = mysql.connector.connect(
    host="localhost",
    user="json_user",
    password="GantiPasswordKuat",
    database="json_lab"
)

cursor = conn.cursor()

cursor.execute("SELECT VERSION()")
version = cursor.fetchone()

print("Versi MySQL:", version[0])

cursor.close()
conn.close()
```

Penjelasan bagian penting:

- `import mysql.connector`: memanggil pustaka koneksi MySQL.
- `mysql.connector.connect(...)`: membuka koneksi database.
- `host`: lokasi server database.
- `user`: user database.
- `password`: password database.
- `database`: nama database yang dipakai.
- `conn.cursor()`: membuat cursor untuk menjalankan SQL.
- `cursor.execute(...)`: menjalankan perintah SQL.
- `fetchone()`: mengambil satu baris hasil.
- `cursor.close()`: menutup cursor.
- `conn.close()`: menutup koneksi.

Jalankan:

```
python db_test.py
```

4.5 Cursor Biasa dan Cursor Dictionary

Cursor biasa mengembalikan hasil sebagai tuple.

```
cursor = conn.cursor()
cursor.execute("SELECT id, nim, nama FROM mahasiswa_profile")
rows = cursor.fetchall()

for row in rows:
    print(row[0], row[1], row[2])
```

Hasilnya dibaca berdasarkan posisi:

```
id, nim, nama
```

Cursor dictionary mengembalikan hasil sebagai dictionary.

```
cursor = conn.cursor(dictionary=True)
cursor.execute("SELECT id, nim, nama FROM mahasiswa_profile")
rows = cursor.fetchall()

for row in rows:
    print(row["id"], row["nim"], row["nama"])
```

Cursor dictionary lebih mudah dibaca karena memakai nama kolom.

Dokumentasi MySQL Connector/Python menunjukkan bahwa cursor dapat dibuat dengan `dictionary=True` agar hasil dikembalikan sebagai dictionary; dokumentasi yang sama juga menunjukkan `prepared=True` untuk prepared cursor. (MySQL Developer Zone)

Saran praktik:

- Gunakan cursor biasa untuk latihan sederhana.
- Gunakan cursor dictionary untuk data yang akan diekspor ke JSON.
- Gunakan prepared cursor untuk query dengan input dari luar.

4.6 Prepared Statement untuk Mengurangi Risiko SQL Injection

SQL injection terjadi ketika input pengguna disisipkan langsung ke string SQL. Ini berbahaya.

Contoh buruk:

```
nim = "2026001"
sql = "SELECT * FROM mahasiswa_profile WHERE nim = '" + nim +
"'"
cursor.execute(sql)
```

Masalahnya: jika `nim` berasal dari input luar, isi input dapat memengaruhi SQL.

Gunakan parameter:

```
nim = "2026001"
sql = "SELECT * FROM mahasiswa_profile WHERE nim = %s"
cursor.execute(sql, (nim,))
```

Lebih aman lagi, gunakan prepared cursor:

```
cursor = conn.cursor(prepared=True)

sql = "SELECT nim, nama FROM mahasiswa_profile WHERE nim =
%s"
cursor.execute(sql, ("2026001",))

for row in cursor.fetchall():
    print(row)
```

Dokumentasi Connector/Python menjelaskan bahwa cursor prepared dapat dibuat dengan `cursor(prepared=True)`, dan cursor ini menggunakan `MySQLCursorPrepared` untuk menjalankan *prepared statements* melalui protokol biner. (MySQL Developer Zone)

Prinsip aman: data input jangan digabung langsung ke string SQL.

4.7 Konversi Python dict dan list ke JSON

Python memakai struktur data:

- `dict` untuk object JSON;
- `list` untuk array JSON;
- `str` untuk string;
- `int` dan `float` untuk angka;
- `True` dan `False` untuk boolean;
- `None` untuk null.

Contoh:

```
import json

profile = {
    "prodi": "Informatika",
    "angkatan": 2026,
    "alamat": {
        "kota": "Tangerang Selatan",
        "provinsi": "Banten"
    },
    "skill": ["Python", "SQL", "Linux"]
}

json_text = json.dumps(profile, ensure_ascii=False)

print(json_text)
```

Penjelasan:

- `json.dumps(...)`: mengubah `dict` atau `list` menjadi string JSON.
- `ensure_ascii=False`: menjaga karakter non-ASCII tetap terbaca alami.
- `profile`: struktur data Python.
- `json_text`: teks JSON siap disimpan ke MySQL.

Membaca JSON string menjadi Python dictionary:

```
data = json.loads(json_text)
print(data["alamat"]["kota"])
```

4.8 Menyiapkan Tabel mahasiswa_profile

Gunakan tabel dari bab sebelumnya. Jika belum ada, buat ulang.

```
CREATE TABLE IF NOT EXISTS mahasiswa_profile (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL UNIQUE,
  nama VARCHAR(100) NOT NULL,
  profile JSON NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Kenapa `nim` dibuat `UNIQUE`?

Karena satu NIM sebaiknya hanya muncul sekali. Ini mencegah duplikasi data mahasiswa.

MySQL mendukung tipe data JSON dan melakukan validasi dokumen JSON yang disimpan ke kolom JSON. Jika dokumen tidak valid, database akan menolak input tersebut. (MySQL Developer Zone)

4.9 Membuat File students.json

Buat file `students.json`.

```
[
  {
    "nim": "2026001",
    "nama": "Alya Pratama",
    "profile": {
      "prodi": "Informatika",
      "angkatan": 2026,
      "alamat": {
        "kota": "Tangerang Selatan",
        "provinsi": "Banten"
      },
      "skill": ["Python", "SQL", "Linux"],
      "minat": ["database", "backend"]
    }
  },
  {
    "nim": "2026002",
    "nama": "Bima Santoso",
    "profile": {
      "prodi": "Sistem Informasi",
      "angkatan": 2026,
      "alamat": {
        "kota": "Depok",
        "provinsi": "Jawa Barat"
      },
      "skill": ["Python", "MySQL", "n8n"],
      "minat": ["api", "automation"]
    }
  },
  {
    "nim": "2026003",
    "nama": "Citra Lestari",
    "profile": {
      "prodi": "Teknik Informatika",
      "angkatan": 2025,
      "alamat": {
        "kota": "Bogor",
        "provinsi": "Jawa Barat"
      },
      "skill": ["SQL", "Python"],
      "minat": ["database", "data"]
    }
  }
]
```

Struktur file:

- root adalah array;
- setiap item adalah object mahasiswa;
- field wajib: `nim`, `nama`, `profile`;
- `profile` berisi object JSON fleksibel.

4.10 Membaca File JSON dengan Python

Buat file `read_students.py`.

```
import json

with open("students.json", "r", encoding="utf-8") as file:
    students = json.load(file)

print("Jumlah data:", len(students))

for student in students:
    print(student["nim"], student["nama"])
```

Penjelasan:

- `open(...)`: membuka file.
- `"r"`: mode baca.
- `encoding="utf-8"`: menjaga karakter teks tetap aman.
- `json.load(file)`: membaca file JSON menjadi struktur Python.
- `len(students)`: menghitung jumlah data.
- `for student in students`: membaca item satu per satu.

Jalankan:

```
python read_students.py
```

4.11 Insert Banyak Data JSON ke MySQL

Buat file `import_students.py`.

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

def load_students(filename):
    with open(filename, "r", encoding="utf-8") as file:
        return json.load(file)

def insert_students(students):
    conn = None

    try:
        conn = mysql.connector.connect(**DB_CONFIG)
        cursor = conn.cursor(prepared=True)

        sql = """
            INSERT INTO mahasiswa_profile (nim, nama,
profile)
            VALUES (%s, %s, %s)
        """

        for student in students:
            nim = student["nim"]
            nama = student["nama"]
            profile_json = json.dumps(student["profile"],
ensure_ascii=False)

            cursor.execute(sql, (nim, nama, profile_json))

        conn.commit()
        print("Import berhasil:", len(students), "data")

    except Error as err:
        if conn:
            conn.rollback()
        print("Error database:", err)
```

```
        except KeyError as err:
            if conn:
                conn.rollback()
            print("Field wajib tidak ada:", err)

        finally:
            if conn and conn.is_connected():
                cursor.close()
                conn.close()

if __name__ == "__main__":
    data = load_students("students.json")
    insert_students(data)
```

Penjelasan struktur:

- `DB_CONFIG`: konfigurasi koneksi database.
- `load_students`: fungsi untuk membaca file JSON.
- `insert_students`: fungsi untuk insert ke database.
- `cursor(prepared=True)`: memakai prepared cursor.
- `%s`: placeholder parameter.
- `json.dumps`: mengubah Python dict menjadi string JSON.
- `conn.commit()`: menyimpan transaksi.
- `conn.rollback()`: membatalkan jika terjadi error.
- `finally`: memastikan koneksi ditutup.
- `if __name__ == "__main__"`: menjalankan script hanya saat file dieksekusi langsung.

MySQL mendukung transaksi dengan `START TRANSACTION`, `COMMIT`, dan `ROLLBACK`; dokumentasi MySQL juga menekankan `START TRANSACTION` sebagai sintaks SQL standar untuk memulai transaksi ad-hoc. (MySQL Developer Zone)

4.12 Batch Insert dengan executemany

Untuk banyak data, lebih rapi memakai `executemany`.

Buat file `import_students_batch.py`.

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

def main():
    conn = None

    try:
        with open("students.json", "r", encoding="utf-8") as
file:
            students = json.load(file)

        rows = []

        for student in students:
            rows.append((
                student["nim"],
                student["nama"],
                json.dumps(student["profile"],
ensure_ascii=False)
            ))

        conn = mysql.connector.connect(**DB_CONFIG)
        cursor = conn.cursor(prepared=True)

        sql = """
            INSERT INTO mahasiswa_profile (nim, nama,
profile)
            VALUES (%s, %s, %s)
        """

        cursor.executemany(sql, rows)
        conn.commit()
```

```
            print("Batch insert berhasil:", cursor.rowcount,
"baris")

        except Error as err:
            if conn:
                conn.rollback()
            print("Error database:", err)

        except json.JSONDecodeError as err:
            print("JSON invalid:", err)

        except KeyError as err:
            print("Field wajib hilang:", err)

        finally:
            if conn and conn.is_connected():
                cursor.close()
                conn.close()

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

Catatan praktis: `executemany` berguna untuk import banyak baris, tetapi validasi tetap harus dilakukan sebelum insert.

4.13 Query Field JSON dari Python

Buat file `query_students.py`.

```
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

try:
    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor(dictionary=True)

    sql = """
        SELECT
            nim,
            nama,
            profile->>'$.prodi' AS prodi,
            profile->>'$.alamat.kota' AS kota,
            profile->>'$.angkatan' AS angkatan
        FROM mahasiswa_profile
        ORDER BY nim
    """

    cursor.execute(sql)
    rows = cursor.fetchall()

    for row in rows:
        print(row)

except Error as err:
    print("Error database:", err)

finally:
    if conn and conn.is_connected():
        cursor.close()
        conn.close()
```

Contoh output:

```
{'nim': '2026001', 'nama': 'Alya Pratama', 'prodi': 'Informatika',
'kota': 'Tangerang Selatan', 'angkatan': '2026'}
```

Dokumentasi MySQL menjelaskan bahwa operator `->` adalah alias `JSON_EXTRACT`, sedangkan `->>` mengembalikan nilai yang sudah di-*unquote*. (MySQL Developer Zone)

4.14 Query dengan Parameter dari Python

Contoh query berdasarkan kota.

Buat file `query_by_city.py`.

```
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

kota_dicari = "Depok"

try:
    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor(dictionary=True, prepared=True)

    sql = """
        SELECT
            nim,
            nama,
            profile->>'$.prodi' AS prodi,
            profile->>'$.alamat.kota' AS kota
        FROM mahasiswa_profile
        WHERE profile->>'$.alamat.kota' = %s
    """

    cursor.execute(sql, (kota_dicari,))
    rows = cursor.fetchall()

    for row in rows:
        print(row)

except Error as err:
    print("Error database:", err)

finally:
    if conn and conn.is_connected():
        cursor.close()
        conn.close()
```

Catatan: gunakan parameter `%s`, bukan menyusun SQL dengan penggabungan string.

4.15 Export Hasil Query Menjadi result.json

Buat file `export_result.py`.

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

try:
    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor(dictionary=True)

    sql = """
        SELECT
            nim,
            nama,
            profile->>'$.prodi' AS prodi,
            profile->>'$.alamat.kota' AS kota,
            profile->'$.skill' AS skill_json
        FROM mahasiswa_profile
        ORDER BY nim
    """

    cursor.execute(sql)
    rows = cursor.fetchall()

    result = []

    for row in rows:
        result.append({
            "nim": row["nim"],
            "nama": row["nama"],
            "prodi": row["prodi"],
            "kota": row["kota"],
            "skill": json.loads(row["skill_json"]) if
row["skill_json"] else []
        })

    with open("result.json", "w", encoding="utf-8") as file:
        json.dump(result, file, ensure_ascii=False, indent=2)
```

```
        print("Export selesai: result.json")

    except Error as err:
        print("Error database:", err)

    finally:
        if conn and conn.is_connected():
            cursor.close()
            conn.close()
```

Penjelasan:

- `cursor(dictionary=True)`: hasil query mudah diolah sebagai dictionary.
- `profile->'$.skill'`: mengambil array JSON.
- `json.loads(...)`: mengubah string JSON array menjadi list Python.
- `json.dump(...)`: menulis hasil ke file JSON.
- `indent=2`: membuat output rapi.
- `ensure_ascii=False`: karakter tetap terbaca baik.

4.16 Menangani Error Koneksi

Error koneksi bisa terjadi karena:

- service database belum aktif;
- user salah;
- password salah;
- database belum dibuat;
- port tidak terbuka;
- host salah.

Contoh penanganan:

```
import mysql.connector
from mysql.connector import Error

try:
    conn = mysql.connector.connect(
        host="localhost",
        user="json_user",
        password="PasswordSalah",
        database="json_lab"
    )

    if conn.is_connected():
        print("Koneksi berhasil")

except Error as err:
    print("Koneksi gagal:", err)

finally:
    if "conn" in locals() and conn.is_connected():
        conn.close()
```

Prinsip: jangan biarkan error tampil mentah tanpa pesan yang bisa dipahami.

4.17 Menangani Error SQL

Error SQL bisa terjadi karena:

- nama tabel salah;
- nama kolom salah;
- query salah;
- constraint gagal;
- data duplikat;
- tipe data tidak cocok;
- JSON invalid.

Contoh:

```
try:
    cursor.execute("SELECT kolom_tidak_ada FROM
mahasiswa_profile")
except Error as err:
    print("Terjadi error SQL:", err)
```

Untuk operasi penting, gunakan rollback:

```
try:
    cursor.execute(sql, data)
    conn.commit()
except Error as err:
    conn.rollback()
    print("Transaksi dibatalkan:", err)
```

Dokumentasi MySQL menyatakan bahwa transaksi lokal didukung melalui `SET autocommit`, `START TRANSACTION`, `COMMIT`, dan `ROLLBACK`. (MySQL Developer Zone)

4.18 Menangani JSON Invalid

Contoh file rusak `students_invalid.json`:

```
[
  {
    "nim": "2026004",
    "nama": "Deni Prakoso",
    "profile": {
      "prodi": "Informatika",
      "angkatan": 2026,
    }
  }
]
```

Ada koma berlebih setelah `2026`.

Script validasi:

```
import json

try:
    with open("students_invalid.json", "r", encoding="utf-8")
as file:
        data = json.load(file)

    print("JSON valid")

except json.JSONDecodeError as err:
    print("JSON invalid")
    print("Pesan:", err.msg)
    print("Baris:", err.lineno)
    print("Kolom:", err.colno)
```

Output akan menunjukkan lokasi error.

Prinsip penting: validasi file sebelum koneksi database. Jangan mengirim data rusak ke MySQL.

4.19 Validasi Struktur Dasar Sebelum Insert

JSON valid secara sintaks belum tentu memiliki field yang dibutuhkan.

Contoh valid tetapi tidak lengkap:

```
{
  "nim": "2026005",
  "profile": {
    "prodi": "Informatika"
  }
}
```

Field `nama` hilang.

Validasi sederhana:

```
def validate_student(student):
    required_fields = ["nim", "nama", "profile"]

    for field in required_fields:
        if field not in student:
            raise ValueError(f"Field wajib hilang: {field}")

    if not isinstance(student["profile"], dict):
        raise ValueError("profile harus berupa object JSON")

    return True
```

Gunakan sebelum insert:

```
for student in students:
    validate_student(student)
```

Catatan: validasi sederhana ini cukup untuk Bab 4. Validasi lebih formal dengan **JSON Schema** dibahas pada Bab 8.

4.20 Transaction dan Rollback Sederhana

Transaksi penting saat insert banyak data. Jika satu data gagal, kita bisa membatalkan semua agar database tidak setengah jadi.

Contoh:

```
try:
    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor(prepared=True)

    conn.start_transaction()

    for student in students:
        validate_student(student)

        cursor.execute(
            """
            INSERT INTO mahasiswa_profile (nim, nama,
profile)
            VALUES (%s, %s, %s)
            """,
            (
                student["nim"],
                student["nama"],
                json.dumps(student["profile"],
ensure_ascii=False)
            )
        )

    conn.commit()
    print("Semua data berhasil disimpan")

except Exception as err:
    if conn:
        conn.rollback()
    print("Import gagal, semua perubahan dibatalkan:", err)
```

Dokumentasi InnoDB menjelaskan bahwa jika autocommit tetap aktif, transaksi dapat dimulai dengan `START TRANSACTION` dan diakhiri dengan `COMMIT` atau `ROLLBACK`. (MySQL Developer Zone)

Pesan kritis: untuk import banyak data, jangan biarkan database terisi separuh.

Hands-on Utama Bab 4 — Importer JSON Mahasiswa

4.21 Struktur Folder Praktik

Buat folder:

```
mkdir json-mysql-python
cd json-mysql-python
```

Struktur akhir:

```
json-mysql-python/
├── students.json
├── importer.py
├── query_students.py
├── export_result.py
└── result.json
```

4.22 Script Lengkap importer.py

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

def read_json_file(filename):
    with open(filename, "r", encoding="utf-8") as file:
        return json.load(file)

def validate_student(student):
    required_fields = ["nim", "nama", "profile"]

    for field in required_fields:
        if field not in student:
            raise ValueError(f"Field wajib hilang: {field}")

    if not isinstance(student["profile"], dict):
        raise ValueError("profile harus berupa object JSON")

    if "prodi" not in student["profile"]:
        raise ValueError("profile.prodi wajib ada")

    return True

def import_students(filename):
    conn = None
    cursor = None

    try:
        students = read_json_file(filename)

        conn = mysql.connector.connect(**DB_CONFIG)
        cursor = conn.cursor(prepared=True)

        conn.start_transaction()

        sql = """
            INSERT INTO mahasiswa_profile (nim, nama,
profile)
            VALUES (%s, %s, %s)
```

```
        """

        for student in students:
            validate_student(student)

            data = (
                student["nim"],
                student["nama"],
                json.dumps(student["profile"],
ensure_ascii=False)
            )

            cursor.execute(sql, data)

        conn.commit()
        print(f"Import berhasil: {len(students)} data")

    except json.JSONDecodeError as err:
        print("JSON invalid")
        print("Pesan:", err.msg)
        print("Baris:", err.lineno)
        print("Kolom:", err.colno)

    except Error as err:
        if conn:
            conn.rollback()
        print("Error database:", err)

    except Exception as err:
        if conn:
            conn.rollback()
        print("Import gagal:", err)

    finally:
        if cursor:
            cursor.close()
        if conn and conn.is_connected():
            conn.close()

if __name__ == "__main__":
    import_students("students.json")
```

4.23 Penjelasan Script importer.py

Bagian import:

```
import json
import mysql.connector
from mysql.connector import Error
```

Artinya:

- `json`: modul bawaan Python untuk membaca dan menulis JSON.
- `mysql.connector`: pustaka koneksi ke MySQL.
- `Error`: class error dari connector untuk menangani error database.

Bagian konfigurasi:

```
DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}
```

Artinya:

- semua konfigurasi koneksi dikumpulkan di satu tempat;
- lebih mudah diganti;
- lebih rapi dibanding menulis ulang di banyak fungsi.

Bagian validasi:

```
required_fields = ["nim", "nama", "profile"]
```

Artinya setiap data mahasiswa wajib punya:

- `nim`;
- `nama`;
- `profile`.

Bagian transaksi:

```
conn.start_transaction()
```

Artinya operasi import dimulai sebagai satu transaksi.

Bagian simpan perubahan:

```
conn.commit()
```

Artinya semua perubahan disimpan permanen.

Bagian batalkan perubahan:

```
conn.rollback()
```

Artinya jika ada error, perubahan dibatalkan.

4.24 Query dari Python dan Export JSON

Buat file `export_result.py`.

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

def main():
    conn = None
    cursor = None

    try:
        conn = mysql.connector.connect(**DB_CONFIG)
        cursor = conn.cursor(dictionary=True)

        sql = """
            SELECT
                nim,
                nama,
                profile->>'$.prodi' AS prodi,
                profile->>'$.alamat.kota' AS kota,
                profile->'$.skill' AS skill
            FROM mahasiswa_profile
            ORDER BY nim
        """

        cursor.execute(sql)
        rows = cursor.fetchall()

        result = []

        for row in rows:
            result.append({
                "nim": row["nim"],
                "nama": row["nama"],
                "prodi": row["prodi"],
                "kota": row["kota"],
                "skill": json.loads(row["skill"]) if
row["skill"] else []
            })
```

```
            with open("result.json", "w", encoding="utf-8") as
    file:
                json.dump(result, file, ensure_ascii=False,
    indent=2)

            print("Export berhasil ke result.json")

        except Error as err:
            print("Error database:", err)

        finally:
            if cursor:
                cursor.close()
            if conn and conn.is_connected():
                conn.close()

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

Jalankan:

```
python export_result.py
```

Cek hasil:

```
cat result.json
```

Atau rapikan dengan Python:

```
python -m json.tool result.json
```

Mini-Lab Bab 4 — Importer JSON Mahasiswa

Tujuan

Mahasiswa membuat importer sederhana yang membaca `students.json`, memvalidasi struktur dasar, menyimpan data ke MySQL, lalu mengekspor ringkasan menjadi `result.json`.

Tugas Praktik

Kerjakan:

- Buat database `json_lab`.
- Buat tabel `mahasiswa_profile`.
- Buat file `students.json`.
- Buat script `importer.py`.
- Gunakan `prepared=True`.
- Gunakan `json.dumps()` untuk menyimpan `profile`.
- Gunakan transaksi.
- Gunakan `rollback` saat error.
- Query field JSON dari Python.
- Export hasil query ke `result.json`.
- Uji dengan JSON valid.
- Uji dengan JSON invalid.
- Uji dengan field wajib yang hilang.

Output yang Diharapkan

Isi tabel:

```
2026001 | Alya Pratama | Informatika | Tangerang Selatan
2026002 | Bima Santoso | Sistem Informasi | Depok
2026003 | Citra Lestari | Teknik Informatika | Bogor
```

Isi `result.json`:

```
[
  {
    "nim": "2026001",
    "nama": "Alya Pratama",
    "prodi": "Informatika",
    "kota": "Tangerang Selatan",
    "skill": ["Python", "SQL", "Linux"]
  }
]
```

Kesalahan Umum Mahasiswa

Hindari:

- Menyimpan Python `dict` langsung ke MySQL tanpa `json.dumps`.
- Menggabungkan input langsung ke string SQL.
- Tidak memakai parameter `%s`.
- Tidak menutup koneksi.
- Tidak memakai `rollback` saat import gagal.
- Tidak menangani `json.JSONDecodeError`.
- Tidak mengecek field wajib.
- Menganggap JSON valid berarti data sudah benar.
- Menyimpan password di script yang dibagikan publik.
- Tidak membedakan error koneksi, error SQL, dan error JSON.

Checklist Akhir Bab

Mahasiswa dinyatakan memahami Bab 4 jika mampu:

- Menghubungkan Python ke MySQL.
- Menggunakan `mysql-connector-python`.
- Membuat cursor biasa.
- Membuat cursor dictionary.
- Membuat prepared cursor.
- Membaca file JSON dengan `json.load`.
- Mengubah `dict` menjadi JSON string dengan `json.dumps`.
- Insert data JSON ke kolom MySQL.
- Query field JSON dari Python.
- Export hasil query dengan `json.dump`.
- Menangani error koneksi.
- Menangani error SQL.
- Menangani JSON invalid.
- Menggunakan transaksi dan `rollback`.
- Menjelaskan risiko *SQL injection*.
- Membuat importer JSON mahasiswa.

Ringkasan Bab

Bab ini adalah inti praktik pemrograman. Mahasiswa belajar menjadikan Python sebagai jembatan antara file JSON dan MySQL. Data dibaca dari `students.json`, divalidasi secara dasar, dikonversi dengan `json.dumps`, disimpan memakai prepared statement, lalu diambil kembali dengan query JSON dan diekspor menjadi `result.json`. Fokus keamanan mulai masuk melalui parameter query, prepared cursor, transaksi, dan rollback. **Pelajaran paling penting: pipeline data yang baik bukan hanya bisa jalan, tetapi harus aman, rapi, bisa gagal dengan terkendali, dan bisa diaudit.**

Bab 5 — SQLAlchemy: Interaksi Database yang Lebih Rapi dan Terstruktur

> **SQLAlchemy membuat interaksi database lebih rapi, tetapi tidak boleh membuat mahasiswa buta SQL. ORM membantu struktur kode; SQL tetap fondasi berpikirnya.**

Bab 5 memperkenalkan SQLAlchemy sebagai pendekatan lebih profesional, membahas *engine*, *connection*, *session*, *model*, tipe JSON, insert object Python ke kolom JSON, query JSON, update nested JSON, dan perbandingan dengan SQL mentah.

Tujuan Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa mampu:

- Menjelaskan mengapa **SQLAlchemy penting** dalam proyek Python.
- Membedakan **SQL mentah**, SQLAlchemy Core, dan ORM.
- Memahami konsep *engine*, *connection*, *session*, dan *model*.
- Membuat model `UserProfile`.
- Menggunakan tipe data `JSON` di SQLAlchemy.
- Insert object Python ke kolom JSON.
- Query field JSON dari SQLAlchemy.
- Update *nested JSON* secara aman.
- Export hasil query ke format JSON API-style.
- Menentukan kapan memakai `mysql-connector-python` dan kapan memakai SQLAlchemy.
- Menjelaskan risiko ORM jika tidak memahami SQL.

5.1 Mengapa SQLAlchemy Penting?

Pada Bab 4, mahasiswa memakai `mysql-connector-python`. Cara itu bagus untuk memahami SQL secara langsung. Namun, ketika proyek membesar, script bisa menjadi panjang dan berulang.

Masalah yang sering muncul:

- koneksi database ditulis berulang;
- SQL tersebar di banyak file;
- struktur tabel tidak terdokumentasi dalam kode;
- hasil query harus diubah manual menjadi object;
- transaksi lebih sulit dikelola secara rapi;
- perubahan skema membuat banyak script harus diedit.

SQLAlchemy membantu membuat interaksi database lebih terstruktur. SQLAlchemy menyediakan dua pendekatan utama:

- **Core**, yaitu cara menulis ekspresi SQL dengan struktur Python.
- **ORM**, yaitu cara memetakan tabel database menjadi class Python.

Dokumentasi SQLAlchemy menyebut SQLAlchemy Unified Tutorial sebagai pengantar terpadu untuk Core dan ORM, dengan gaya kerja modern yang memakai `select()` dan semantik transaksi yang selaras antara Core dan ORM. (SQLAlchemy)

Makna praktis: SQLAlchemy membuat kode database lebih rapi, tetapi mahasiswa tetap harus memahami SQL di baliknya.

5.2 Perbedaan SQL Mentah dan ORM

SQL mentah berarti kita menulis query langsung.

Contoh:

```
cursor.execute(
    "SELECT nim, nama FROM mahasiswa_profile WHERE nim = %s",
    ("2026001",)
)
```

ORM berarti kita membuat class Python yang mewakili tabel.

Contoh konsep:

```
user = UserProfile(nim="2026001", nama="Alya Pratama")
session.add(user)
session.commit()
```

Perbandingan ringkas:

Pendekatan	Kelebihan	Risiko
SQL mentah	jelas, langsung, dekat dengan database	kode panjang jika proyek besar
SQLAlchemy Core	lebih rapi, tetap dekat dengan SQL	perlu belajar sintaks ekspresi
SQLAlchemy ORM	cocok untuk aplikasi terstruktur	bisa membuat mahasiswa lupa SQL
Campuran	fleksibel untuk proyek nyata	harus disiplin desain

Sikap terbaik: gunakan ORM untuk struktur, gunakan SQL mentah saat perlu kontrol penuh.

5.3 Engine, Connection, Session, dan Model

SQLAlchemy memiliki beberapa konsep penting.

Engine

Engine adalah pintu utama ke database.

```
engine =
create_engine("mysql+mysqlconnector://user:password@loca
lhost/json_lab")
```

Engine menyimpan informasi koneksi dan cara SQLAlchemy berbicara dengan database.

Connection

Connection adalah koneksi aktif untuk menjalankan perintah SQL.

```
with engine.connect() as conn:
    result = conn.execute(text("SELECT VERSION()"))\
```

Session

Session adalah unit kerja ORM. Ia mengelola object, transaksi, insert, update, delete, dan commit.

```
with Session(engine) as session:
    session.add(user)
    session.commit()
```

Dokumentasi SQLAlchemy menjelaskan `sessionmaker` sebagai factory untuk membuat object `Session` berdasarkan konfigurasi tertentu. (SQLAlchemy)

Model

Model adalah class Python yang mewakili tabel database.

```
class UserProfile(Base):
    __tablename__ = "user_profiles"
```

Model membuat struktur tabel terlihat jelas dalam kode.

5.4 Instalasi SQLAlchemy

Aktifkan *virtual environment*:

```
source venv/bin/activate
```

Install package:

```
pip install SQLAlchemy mysql-connector-python
```

Cek versi:

```
python -c "import sqlalchemy; print(sqlalchemy.__version__)"
```

Catatan: `mysql-connector-python` tetap dipakai sebagai driver koneksi. SQLAlchemy berada di atasnya sebagai lapisan abstraksi.

5.5 Membuat Koneksi SQLAlchemy

Buat file `db.py`.

```
from sqlalchemy import create_engine

DB_URL =
"mysql+mysqlconnector://json_user:GantiPasswordKuat@localhost
/json_lab"

engine = create_engine(
    DB_URL,
    echo=True
)
```

Penjelasan:

- `create_engine`: membuat engine SQLAlchemy.
- `mysql+mysqlconnector`: memakai MySQL dengan driver `mysql-connector-python`.
- `json_user`: user database.
- `GantiPasswordKuat`: password database.
- `localhost`: lokasi database.
- `json_lab`: nama database.
- `echo=True`: menampilkan SQL yang dijalankan SQLAlchemy.

Untuk pembelajaran, `echo=True` sangat berguna. Mahasiswa bisa melihat SQL yang dihasilkan ORM.

Prinsip belajar: jangan percaya ORM secara buta; lihat SQL yang dibuat.

5.6 Membuat Model UserProfile

Buat file `models.py`.

```
from sqlalchemy import Integer, String, JSON, DateTime, func
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
    pass

class UserProfile(Base):
    __tablename__ = "user_profiles"

    id: Mapped[int] = mapped_column(Integer, primary_key=True,
autoincrement=True)
    nim: Mapped[str] = mapped_column(String(20), unique=True,
nullable=False)
    nama: Mapped[str] = mapped_column(String(100), nullable=False)
    profile: Mapped[dict] = mapped_column(JSON, nullable=False)
    created_at: Mapped[str] = mapped_column(DateTime,
server_default=func.now())
```

Penjelasan:

- `DeclarativeBase`: fondasi untuk model ORM.
- `UserProfile`: class Python yang mewakili tabel.
- `__tablename__`: nama tabel di database.
- `Mapped[int]`: anotasi tipe data untuk kolom.
- `mapped_column`: mendefinisikan kolom.
- `Integer`: tipe angka.
- `String(20)`: teks maksimal 20 karakter.
- `JSON`: tipe JSON.
- `unique=True`: nilai harus unik.
- `nullable=False`: wajib diisi.
- `server_default=func.now()`: waktu dibuat diisi oleh database.

SQLAlchemy mendukung tipe JSON generik dan berbagai operasi indeks JSON. Pada MySQL, ekspresi JSON tertentu dapat diterjemahkan menjadi fungsi seperti `JSON_EXTRACT`. Dokumentasi SQLAlchemy menunjukkan bahwa pada MySQL, penggunaan ekspresi JSON dapat memunculkan fungsi `JSON_EXTRACT`. (SQLAlchemy)

5.7 Membuat Tabel dari Model

Buat file `create_tables.py`.

```
from db import engine
from models import Base

Base.metadata.create_all(engine)

print("Tabel siap")
```

Jalankan:

```
python create_tables.py
```

Jika `echo=True`, mahasiswa akan melihat SQL `CREATE TABLE` yang dijalankan.

Catatan kritis: untuk proyek produksi, perubahan skema sebaiknya dikelola dengan tool migrasi. Untuk bab ini, `create_all()` cukup untuk latihan.

5.8 Insert Object Python ke Kolom JSON

Buat file `insert_user.py`.

```
from sqlalchemy.orm import Session
from db import engine
from models import UserProfile


profile_data = {
    "prodi": "Informatika",
    "angkatan": 2026,
    "alamat": {
        "kota": "Tangerang Selatan",
        "provinsi": "Banten"
    },
    "skill": ["Python", "SQL", "Linux"],
    "minat": ["database", "backend"]
}

user = UserProfile(
    nim="2026001",
    nama="Alya Pratama",
    profile=profile_data
)

with Session(engine) as session:
    session.add(user)
    session.commit()

print("Data user berhasil disimpan")
```

Penjelasan:

- `profile_data` adalah Python `dict`.
- SQLAlchemy mengirim `dict` tersebut ke kolom JSON.
- `session.add(user)` menandai object untuk disimpan.
- `session.commit()` menjalankan transaksi.

Kekuatan SQLAlchemy: mahasiswa bisa bekerja dengan object Python, tetapi database tetap menyimpan sebagai JSON.

5.9 Insert Banyak User

Buat file `insert_many_users.py`.

```
from sqlalchemy.orm import Session
from db import engine
from models import UserProfile


users = [
    UserProfile(
        nim="2026002",
        nama="Bima Santoso",
        profile={
            "prodi": "Sistem Informasi",
            "angkatan": 2026,
            "alamat": {
                "kota": "Depok",
                "provinsi": "Jawa Barat"
            },
            "skill": ["Python", "MySQL", "n8n"],
            "minat": ["api", "automation"]
        }
    ),
    UserProfile(
        nim="2026003",
        nama="Citra Lestari",
        profile={
            "prodi": "Teknik Informatika",
            "angkatan": 2025,
            "alamat": {
                "kota": "Bogor",
                "provinsi": "Jawa Barat"
            },
            "skill": ["SQL", "Python"],
            "minat": ["database", "data"]
        }
    )
]

with Session(engine) as session:
    session.add_all(users)
    session.commit()

print("Banyak user berhasil disimpan")
```

Catatan: untuk import ribuan data, perlu strategi batch yang lebih serius. Namun, untuk bab ini, `add_all()` cukup sebagai fondasi.

5.10 Query Data Dasar dengan SQLAlchemy

Buat file `query_users.py`.

```
from sqlalchemy import select
from sqlalchemy.orm import Session
from db import engine
from models import UserProfile

with Session(engine) as session:
    stmt = select(UserProfile).order_by(UserProfile.nim)
    users = session.scalars(stmt).all()

    for user in users:
        print(user.nim, user.nama, user.profile)
```

Penjelasan:

- `select(UserProfile)`: membuat query SELECT.
- `order_by(UserProfile.nim)`: mengurutkan berdasarkan NIM.
- `session.scalars(stmt).all()`: mengambil hasil sebagai object model.
- `user.profile`: berisi dictionary Python dari kolom JSON.

5.11 Query Berdasarkan Field JSON

Query mahasiswa berdasarkan kota.

Buat file `query_by_json.py`.

```
from sqlalchemy import select
from sqlalchemy.orm import Session
from db import engine
from models import UserProfile


with Session(engine) as session:
    stmt = (
        select(UserProfile)

.where(UserProfile.profile["alamat"]["kota"].as_string() ==
"Depok")
    )

    users = session.scalars(stmt).all()

    for user in users:
        print(user.nim, user.nama,
user.profile["alamat"]["kota"])
```

Penjelasan:

- `UserProfile.profile["alamat"]["kota"]`: akses field JSON.
- `.as_string()`: memberi tahu SQLAlchemy bahwa nilai dibandingkan sebagai string.
- `== "Depok"`: filter kota.

Dokumentasi SQLAlchemy merekomendasikan penggunaan bentuk seperti `.as_string()` saat membandingkan elemen JSON sebagai string agar ekspresi lebih jelas lintas dialek. (GitHub)

Query mahasiswa berdasarkan prodi:

```
from sqlalchemy import select
from sqlalchemy.orm import Session
from db import engine
from models import UserProfile


with Session(engine) as session:
    stmt = (
        select(UserProfile.nim, UserProfile.nama)
        .where(UserProfile.profile["prodi"].as_string() ==
"Informatika")
    )

    rows = session.execute(stmt).all()

    for row in rows:
        print(row.nim, row.nama)
```

Catatan kritis: query JSON dengan ORM tetap menghasilkan SQL di database. Karena itu, aktifkan `echo=True` saat belajar agar SQL yang dihasilkan terlihat.

5.12 Update Nested JSON

Update JSON dengan ORM perlu hati-hati. Perubahan langsung pada nested dictionary kadang tidak otomatis terdeteksi sebagai perubahan oleh ORM.

Cara aman untuk latihan: buat salinan dictionary, ubah, lalu assign kembali ke kolom `profile`.

Buat file `update_nested_json.py`.

```
from sqlalchemy import select
from sqlalchemy.orm import Session
from db import engine
from models import UserProfile


with Session(engine) as session:
    stmt = select(UserProfile).where(UserProfile.nim ==
"2026001")
    user = session.scalars(stmt).one()

    new_profile = dict(user.profile)
    new_profile["alamat"] = dict(new_profile["alamat"])
    new_profile["alamat"]["kota"] = "Jakarta Selatan"

    user.profile = new_profile

    session.commit()

print("Nested JSON berhasil diupdate")
```

Penjelasan:

- ambil object user;
- salin `profile`;
- salin nested object `alamat`;
- ubah field `kota`;
- assign ulang `user.profile`;
- commit.

Pesan penting: untuk update nested JSON, jangan asal mengubah object di dalam dictionary tanpa memahami mekanisme tracking ORM.

Untuk proyek lebih lanjut, SQLAlchemy memiliki ekstensi mutable untuk melacak perubahan struktur seperti dictionary. Namun, untuk mahasiswa, cara assign ulang lebih mudah dipahami.

5.13 Update JSON dengan SQL Mentah melalui SQLAlchemy

Kadang cara terbaik adalah memakai SQL MySQL secara langsung, terutama untuk fungsi JSON seperti `JSON_SET`.

Buat file `update_with_raw_sql.py`.

```
from sqlalchemy import text
from db import engine


sql = text("""
    UPDATE user_profiles
    SET profile = JSON_SET(profile, '$.skill[0]',
:skill_baru)
    WHERE nim = :nim
""")

with engine.begin() as conn:
    conn.execute(sql, {
        "skill_baru": "Python Advanced",
        "nim": "2026001"
    })

print("Update JSON_SET berhasil")
```

Penjelasan:

- `text(...)`: menjalankan SQL mentah melalui SQLAlchemy.
- `:skill_baru`: parameter aman.
- `:nim`: parameter aman.
- `engine.begin()`: membuka transaksi otomatis.
- jika sukses, transaksi commit;
- jika gagal, transaksi rollback.

Dokumentasi MySQL menjelaskan operator `->` sebagai alias untuk `JSON_EXTRACT`, dan fungsi JSON MySQL dapat digunakan dalam SQL statement untuk mengambil atau mengubah dokumen JSON. (MySQL Developer Zone)

Prinsip profesional: ORM boleh dipakai, tetapi fungsi database tetap boleh digunakan jika lebih tepat.

5.14 Export Hasil Query ke Format JSON API-style

Format API-style biasanya berbentuk object dengan metadata dan data.

Contoh target:

```
{
  "status": "success",
  "count": 2,
  "data": [
    {
      "nim": "2026001",
      "nama": "Alya Pratama",
      "profile": {
        "prodi": "Informatika"
      }
    }
  ]
}
```

Buat file `export_api_style.py`.

```
import json
from sqlalchemy import select
from sqlalchemy.orm import Session
from db import engine
from models import UserProfile


with Session(engine) as session:
    stmt = select(UserProfile).order_by(UserProfile.nim)
    users = session.scalars(stmt).all()

    data = []

    for user in users:
        data.append({
            "id": user.id,
            "nim": user.nim,
            "nama": user.nama,
            "profile": user.profile
        })

    result = {
        "status": "success",
        "count": len(data),
        "data": data
```

```
    }

with open("users_api_result.json", "w", encoding="utf-8") as
file:
    json.dump(result, file, ensure_ascii=False, indent=2)

print("Export selesai: users_api_result.json")
```

Penjelasan:

- `data`: daftar user.
- `result`: object utama response.
- `status`: status proses.
- `count`: jumlah data.
- `json.dump`: menulis file JSON.
- `indent=2`: membuat JSON rapi.

5.15 Membandingkan SQL Mentah dan SQLAlchemy

SQL mentah dengan connector

```
cursor.execute(
    """
    SELECT nim, nama, profile->>'$.alamat.kota' AS kota
    FROM mahasiswa_profile
    WHERE profile->>'$.alamat.kota' = %s
    """,
    ("Depok",)
)
```

Kelebihan:

- sangat jelas;
- dekat dengan kemampuan MySQL;
- mudah dipakai untuk fungsi JSON spesifik.

Kekurangan:

- mudah berulang;
- struktur tabel tidak terlihat sebagai model;
- kode bisa sulit dirawat jika proyek besar.

SQLAlchemy ORM

```
stmt = (
    select(UserProfile)
    .where(UserProfile.profile["alamat"]["kota"].as_string()
== "Depok")
)
```

Kelebihan:

- kode lebih terstruktur;
- tabel menjadi class;
- object lebih mudah dipakai di aplikasi;
- session mengelola transaksi.

Kekurangan:

- SQL yang dihasilkan tidak selalu terlihat;
- debugging bisa lebih rumit;
- mahasiswa bisa lupa konsep SQL dasar.

Kesimpulan: SQL mentah tajam untuk kontrol. SQLAlchemy kuat untuk struktur.

5.16 Kapan Memakai mysql-connector-python?

Gunakan `mysql-connector-python` jika:

- sedang belajar SQL dasar;
- butuh kontrol penuh terhadap query;
- script kecil dan langsung;
- tugas fokus pada SQL MySQL;
- ingin memahami `JSON_EXTRACT`, `JSON_SET`, `JSON_REMOVE` secara eksplisit;
- membuat tool import/export sederhana;
- ingin minim lapisan abstraksi.

Contoh situasi:

- script import `students.json`;
- query laporan sederhana;
- debugging fungsi JSON;
- latihan transaksi dasar.

5.17 Kapan Memakai SQLAlchemy?

Gunakan SQLAlchemy jika:

- proyek mulai besar;
- banyak tabel dan relasi;
- butuh struktur model;
- kode database dipakai berulang;
- ingin memisahkan logic aplikasi dan akses database;
- butuh transaksi yang lebih rapi;
- ingin membuat backend data yang lebih terorganisir;
- ingin export object ke response JSON API-style.

SQLAlchemy ORM cocok ketika tabel dapat dipetakan menjadi object Python. Dokumentasi SQLAlchemy ORM menjelaskan ORM sebagai lapisan yang membangun SQL tingkat tinggi secara otomatis dan mengelola persistensi object Python. (SQLAlchemy)

Saran pedagogis: belajar SQL mentah dulu, baru pakai ORM. Jangan dibalik.

5.18 Risiko ORM: Jangan Sampai Tidak Paham SQL

ORM sering membuat pemula merasa database itu seperti object biasa. Ini nyaman, tetapi berbahaya.

Risiko ORM:

- tidak tahu query SQL yang dijalankan;
- tidak sadar query menjadi lambat;
- tidak paham join;
- tidak paham transaksi;
- tidak paham index;
- tidak paham `NULL`;
- tidak paham tipe data JSON;
- salah update nested JSON;
- terlalu banyak query kecil;
- sulit debug performa.

Aturan disiplin:

- Aktifkan `echo=True` saat belajar.
- Baca SQL yang dihasilkan.
- Pahami `SELECT`, `INSERT`, `UPDATE`, `DELETE`.
- Pahami `COMMIT` dan `ROLLBACK`.
- Tetap pelajari `EXPLAIN`.
- Jangan memakai ORM untuk menutupi desain database yang buruk.

Buku attached juga menegaskan bahwa Bab 7 nanti akan membahas indexing dan performance, termasuk generated column serta strategi index agar query JSON tidak menjadi lambat.

Hands-on Bab 5

5.19 Struktur Folder Praktik

Buat folder:

```
mkdir sqlalchemy-json-lab
cd sqlalchemy-json-lab
```

Struktur file:

```
sqlalchemy-json-lab/
├── db.py
├── models.py
├── create_tables.py
├── insert_user.py
├── insert_many_users.py
├── query_users.py
├── query_by_json.py
├── update_nested_json.py
├── update_with_raw_sql.py
└── export_api_style.py
```

5.20 File db.py

from sqlalchemy import create_engine

```
DB_URL =
"mysql+mysqlconnector://json_user:GantiPasswordKuat@localhost
/json_lab"

engine = create_engine(
    DB_URL,
    echo=True
)
```

5.21 File models.py

```
from sqlalchemy import Integer, String, JSON, DateTime, func
from sqlalchemy.orm import DeclarativeBase, Mapped,
mapped_column


class Base(DeclarativeBase):
    pass


class UserProfile(Base):
    __tablename__ = "user_profiles"

    id: Mapped[int] = mapped_column(Integer,
primary_key=True, autoincrement=True)
    nim: Mapped[str] = mapped_column(String(20), unique=True,
nullable=False)
    nama: Mapped[str] = mapped_column(String(100),
nullable=False)
    profile: Mapped[dict] = mapped_column(JSON,
nullable=False)
    created_at: Mapped[str] = mapped_column(DateTime,
server_default=func.now())
```

5.22 File create_tables.py

```
from db import engine
from models import Base

Base.metadata.create_all(engine)

print("Tabel berhasil dibuat")
```

Jalankan:

```
python create_tables.py
```

5.23 File insert_user.py

```
from sqlalchemy.orm import Session
from db import engine
from models import UserProfile


user = UserProfile(
    nim="2026001",
    nama="Alya Pratama",
    profile={
        "prodi": "Informatika",
        "angkatan": 2026,
        "alamat": {
            "kota": "Tangerang Selatan",
            "provinsi": "Banten"
        },
        "skill": ["Python", "SQL", "Linux"]
    }
)

with Session(engine) as session:
    session.add(user)
    session.commit()

print("Data user berhasil dimasukkan")
```

5.24 File query_by_json.py

```
from sqlalchemy import select
from sqlalchemy.orm import Session
from db import engine
from models import UserProfile


with Session(engine) as session:
    stmt = (
        select(UserProfile)

.where(UserProfile.profile["alamat"]["kota"].as_string() ==
"Tangerang Selatan")
    )

    users = session.scalars(stmt).all()

    for user in users:
        print(user.nim, user.nama,
user.profile["alamat"]["kota"])
```

5.25 File update_nested_json.py

```
from sqlalchemy import select
from sqlalchemy.orm import Session
from db import engine
from models import UserProfile


with Session(engine) as session:
    stmt = select(UserProfile).where(UserProfile.nim ==
"2026001")
    user = session.scalars(stmt).one()

    new_profile = dict(user.profile)
    new_profile["alamat"] = dict(new_profile["alamat"])
    new_profile["alamat"]["kota"] = "Serpong"
    new_profile["skill"] = list(new_profile["skill"])
    new_profile["skill"].append("SQLAlchemy")

    user.profile = new_profile
    session.commit()

print("Update nested JSON selesai")
```

5.26 File export_api_style.py

```
import json
from sqlalchemy import select
from sqlalchemy.orm import Session
from db import engine
from models import UserProfile


with Session(engine) as session:
    stmt = select(UserProfile).order_by(UserProfile.nim)
    users = session.scalars(stmt).all()

    data = [
        {
            "id": user.id,
            "nim": user.nim,
            "nama": user.nama,
            "profile": user.profile
        }
        for user in users
    ]

    result = {
        "status": "success",
        "count": len(data),
        "data": data
    }

with open("users_api_result.json", "w", encoding="utf-8") as
file:
    json.dump(result, file, ensure_ascii=False, indent=2)

print("Export API-style selesai")
```

Mini-Lab Bab 5 — Backend Data Profil Sederhana

Tujuan

Mahasiswa membuat backend data profil sederhana memakai SQLAlchemy. Data inti disimpan sebagai kolom relasional. Data fleksibel disimpan sebagai JSON.

Tugas Praktik

Kerjakan:

- Buat folder proyek `sqlalchemy-json-lab`.
- Install SQLAlchemy dan driver MySQL.
- Buat `db.py`.
- Buat model `UserProfile`.
- Buat tabel dari model.
- Insert minimal 3 profil user.
- Query semua user.
- Query user berdasarkan `profile.alamat.kota`.
- Query user berdasarkan `profile.prodi`.
- Update `profile.alamat.kota`.
- Tambahkan skill baru ke array `profile.skill`.
- Export hasil ke `users_api_result.json`.
- Bandingkan SQL mentah dan SQLAlchemy.
- Tulis refleksi pendek: kapan ORM membantu, kapan ORM menyulitkan.

Contoh Output API-style

```
{
  "status": "success",
  "count": 3,
  "data": [
    {
      "id": 1,
      "nim": "2026001",
      "nama": "Alya Pratama",
      "profile": {
        "prodi": "Informatika",
        "angkatan": 2026,
        "alamat": {
          "kota": "Serpong",
          "provinsi": "Banten"
        },
        "skill": ["Python", "SQL", "Linux", "SQLAlchemy"]
      }
    }
  ]
}
```

Kesalahan Umum Mahasiswa

Hindari:

- Memakai ORM sebelum paham SQL dasar.
- Tidak mengaktifkan `echo=True` saat belajar.
- Tidak membaca SQL yang dihasilkan SQLAlchemy.
- Mengira ORM selalu lebih cepat.
- Mengira ORM selalu lebih aman.
- Mengubah nested JSON tanpa assign ulang.
- Menyimpan semua data ke JSON.
- Tidak memakai constraint seperti `unique=True`.
- Tidak memahami perbedaan *session* dan *connection*.
- Tidak menutup transaksi dengan benar.
- Tidak membedakan SQLAlchemy Core dan ORM.

Checklist Akhir Bab

Mahasiswa dinyatakan memahami Bab 5 jika mampu:

- Menjelaskan fungsi SQLAlchemy.
- Menjelaskan perbedaan SQL mentah dan ORM.
- Membuat *engine*.
- Membuat model ORM.
- Membuat tabel dari model.
- Menggunakan tipe `JSON`.
- Insert object Python ke kolom JSON.
- Query field JSON dengan SQLAlchemy.
- Update nested JSON.
- Menjalankan SQL mentah melalui SQLAlchemy.
- Export hasil query ke JSON API-style.
- Menjelaskan kapan memakai `mysql-connector-python`.
- Menjelaskan kapan memakai SQLAlchemy.
- Menjelaskan risiko ORM jika tidak paham SQL.

Ringkasan Bab

Bab ini memperkenalkan SQLAlchemy sebagai lapisan kerja database yang lebih rapi dan profesional. Mahasiswa belajar bahwa tabel dapat direpresentasikan sebagai class Python, koneksi dikelola melalui *engine*, transaksi dikelola melalui *session*, dan kolom JSON dapat diisi langsung dari object Python. SQLAlchemy membantu membuat kode lebih terstruktur, tetapi tetap harus digunakan dengan disiplin. **ORM bukan pengganti pemahaman SQL; ORM adalah alat untuk menulis aplikasi database yang lebih rapi ketika fondasi SQL sudah kuat.**

Bab 6 — Query JSON Canggih: Filtering, Aggregation, dan JSON_TABLE

> **JSON tidak boleh berhenti sebagai tempat simpan data; JSON harus bisa dibaca, di filter, diubah menjadi tabel, dan dijadikan laporan yang dapat dipercaya.**

Bab 6 diarahkan untuk membuat mahasiswa “naik kelas” dari sekadar menyimpan JSON menjadi mampu menganalisis, memfilter, menggabungkan, dan mengubah JSON ke bentuk tabular memakai `JSON_TABLE`.

Tujuan Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa mampu:

- Melakukan query pada **nested JSON**.
- Memfilter data berdasarkan **object** dan **array**.
- Menggunakan `JSON_CONTAINS`.
- Menggunakan `JSON_OVERLAPS`.
- Menggunakan `MEMBER OF`.
- Membuat agregasi dengan `JSON_ARRAYAGG`.
- Membuat agregasi dengan `JSON_OBJECTAGG`.
- Mengubah JSON menjadi tabel dengan `JSON_TABLE`.
- Melakukan *join* antara tabel relasional dan data JSON.
- Membuat laporan transaksi dari data JSON.
- Memahami batasan performa query JSON.

6.1 Mengapa Query JSON Canggih Penting?

Pada bab sebelumnya, mahasiswa sudah bisa:

- membuat kolom JSON;
- insert dokumen JSON;
- membaca field JSON;
- update sebagian isi JSON;
- memakai Python dan SQLAlchemy.

Namun, kemampuan itu belum cukup. Sistem nyata tidak hanya menyimpan data. Sistem nyata harus menjawab pertanyaan.

Contoh pertanyaan:

- Customer dari kota mana yang paling banyak melakukan transaksi?
- Produk apa yang paling sering dibeli?
- Order mana yang berisi produk tertentu?
- Customer mana yang membeli beberapa kategori produk?
- Berapa total transaksi per kota?
- Bagaimana mengubah array JSON menjadi baris tabel?

Pertanyaan seperti itu membutuhkan query JSON yang lebih maju.

Dokumentasi MySQL menjelaskan bahwa `JSON_TABLE()` digunakan untuk mengekstrak data dari dokumen JSON dan mengembalikannya sebagai tabel relasional dengan kolom yang ditentukan. (MySQL Developer Zone)

Intinya: JSON yang tidak bisa dianalisis hanya menjadi arsip. JSON yang bisa di-query menjadi aset data.

6.2 Dataset Order Berbasis JSON

Bab ini memakai studi kasus transaksi toko sederhana. Data inti customer disimpan dalam tabel relasional. Detail order disimpan dalam kolom JSON.

Desain ini sengaja dibuat hybrid:

- data customer penting masuk tabel `customers`;
- data order fleksibel masuk tabel `orders_json`;
- item transaksi disimpan sebagai array JSON.

Buku *Unifying Business, Data, and Code* memakai contoh produk dan order untuk menunjukkan bahwa JSON cocok memodelkan data seperti SKU, harga, customer, lokasi pengiriman, item pembelian, dan waktu order.

6.3 Membuat Tabel customers

```
DROP TABLE IF EXISTS orders_json;
DROP TABLE IF EXISTS customers;

CREATE TABLE customers (
  customer_id INT AUTO_INCREMENT PRIMARY KEY,
  nama VARCHAR(100) NOT NULL,
  email VARCHAR(150) NOT NULL UNIQUE,
  kota VARCHAR(100) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Penjelasan:

- `customer_id`: identitas customer.
- `nama`: nama customer.
- `email`: email unik.
- `kota`: kota customer, sengaja dijadikan kolom biasa karena sering difilter.
- `created_at`: waktu data dibuat.

Kenapa `kota` tidak dimasukkan ke JSON?
Karena kota sering dipakai untuk filter dan laporan. Data yang sering dicari sebaiknya tetap menjadi kolom relasional.

6.4 Membuat Tabel orders_json

```
CREATE TABLE orders_json (
  order_id INT AUTO_INCREMENT PRIMARY KEY,
  customer_id INT NOT NULL,
  order_code VARCHAR(30) NOT NULL UNIQUE,
  order_data JSON NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

Penjelasan:

- `order_id`: identitas order.
- `customer_id`: penghubung ke tabel customer.
- `order_code`: kode transaksi.
- `order_data`: detail order dalam JSON.
- `FOREIGN KEY`: menjaga hubungan customer dan order.

Contoh struktur `order_data`:

```
{
  "tanggal": "2026-05-26",
  "status": "paid",
  "payment": {
    "method": "transfer",
    "currency": "IDR"
  },
  "items": [
    {
      "sku": "BK-JSON-001",
      "nama_produk": "Buku JSON Praktis",
      "kategori": "buku",
      "qty": 2,
      "harga": 75000
    }
  ]
}
```

6.5 Insert Dataset Customer

```
INSERT INTO customers (nama, email, kota)
VALUES
  ('Alya Pratama', 'alya@example.test', 'Tangerang Selatan'),
  ('Bima Santoso', 'bima@example.test', 'Depok'),
  ('Citra Lestari', 'citra@example.test', 'Bogor'),
  ('Deni Prakoso', 'deni@example.test', 'Tangerang Selatan');
```

6.6 Insert Dataset Order JSON

```
INSERT INTO orders_json (customer_id, order_code, order_data)
VALUES
(
  1,
  'ORD-2026-001',
  JSON_OBJECT(
    'tanggal', '2026-05-26',
    'status', 'paid',
    'payment', JSON_OBJECT(
      'method', 'transfer',
      'currency', 'IDR'
    ),
    'items', JSON_ARRAY(
      JSON_OBJECT(
        'sku', 'BK-JSON-001',
        'nama_produk', 'Buku JSON Praktis',
        'kategori', 'buku',
        'qty', 2,
        'harga', 75000
      ),
      JSON_OBJECT(
        'sku', 'USB-032',
        'nama_produk', 'Flashdisk 32GB',
        'kategori', 'aksesori',
        'qty', 1,
        'harga', 65000
      )
    )
  )
),
(
  2,
  'ORD-2026-002',
  JSON_OBJECT(
    'tanggal', '2026-05-26',
```

```
    'status', 'paid',
    'payment', JSON_OBJECT(
      'method', 'ewallet',
      'currency', 'IDR'
    ),
    'items', JSON_ARRAY(
      JSON_OBJECT(
        'sku', 'KB-MECH-001',
        'nama_produk', 'Keyboard Mekanik',
        'kategori', 'perangkat',
        'qty', 1,
        'harga', 350000
      ),
      JSON_OBJECT(
        'sku', 'BK-JSON-001',
        'nama_produk', 'Buku JSON Praktis',
        'kategori', 'buku',
        'qty', 1,
        'harga', 75000
      )
    )
  )
),
(
  3,
  'ORD-2026-003',
  JSON_OBJECT(
    'tanggal', '2026-05-27',
    'status', 'pending',
    'payment', JSON_OBJECT(
      'method', 'transfer',
      'currency', 'IDR'
    ),
    'items', JSON_ARRAY(
      JSON_OBJECT(
        'sku', 'MOUSE-001',
        'nama_produk', 'Mouse Optik',
        'kategori', 'perangkat',
        'qty', 2,
        'harga', 90000
      )
    )
  )
),
(
  4,
  'ORD-2026-004',
  JSON_OBJECT(
```

```
    'tanggal', '2026-05-27',
    'status', 'paid',
    'payment', JSON_OBJECT(
      'method', 'cash',
      'currency', 'IDR'
    ),
    'items', JSON_ARRAY(
      JSON_OBJECT(
        'sku', 'BK-DB-001',
        'nama_produk', 'Buku Database Dasar',
        'kategori', 'buku',
        'qty', 1,
        'harga', 80000
      ),
      JSON_OBJECT(
        'sku', 'USB-032',
        'nama_produk', 'Flashdisk 32GB',
        'kategori', 'aksesori',
        'qty', 2,
        'harga', 65000
      )
    )
  )
);
```

6.7 Query Nested JSON

Ambil status order:

```
SELECT
  order_code,
  order_data->>'$.status' AS status
FROM orders_json;
```

Ambil metode pembayaran:

```
SELECT
  order_code,
  order_data->>'$.payment.method' AS metode_bayar
FROM orders_json;
```

Ambil item pertama dalam order:

```
SELECT
  order_code,
  order_data->'$.items[0]' AS item_pertama
FROM orders_json;
```

Ambil SKU item pertama:

```
SELECT
  order_code,
  order_data->>'$.items[0].sku' AS sku_pertama
FROM orders_json;
```

Catatan: array JSON dimulai dari indeks `0`. Jika mengambil item pertama, gunakan `[0]`.

6.8 Filter Berdasarkan Object

Filter order dengan status `paid`:

```
SELECT
  order_code,
  order_data->>'$.status' AS status
FROM orders_json
WHERE order_data->>'$.status' = 'paid';
```

Filter order dengan metode pembayaran `transfer`:

```
SELECT
  order_code,
  order_data->>'$.payment.method' AS metode_bayar
FROM orders_json
WHERE order_data->>'$.payment.method' = 'transfer';
```

Filter order tanggal tertentu:

```
SELECT
  order_code,
  order_data->>'$.tanggal' AS tanggal
FROM orders_json
WHERE order_data->>'$.tanggal' = '2026-05-26';
```

6.9 Filter Berdasarkan Array dengan JSON_CONTAINS

`JSON_CONTAINS` digunakan untuk mengecek apakah dokumen JSON mengandung candidate tertentu. Dokumentasi MySQL menjelaskan format dasarnya sebagai `JSON_CONTAINS(target, candidate[, path])`. (MySQL Developer Zone)

Contoh mencari order yang memiliki item dengan SKU `BK-JSON-001`.

```
SELECT
  order_code,
  JSON_PRETTY(order_data->'$.items') AS items
FROM orders_json
WHERE JSON_CONTAINS(
  order_data->'$.items',
  JSON_OBJECT('sku', 'BK-JSON-001')
);
```

Namun, untuk array object yang memiliki banyak field, pendekatan ini sering sensitif terhadap bentuk object. Cara yang lebih kuat adalah memakai `JSON_TABLE`, yang akan dibahas sebentar lagi.

Contoh `JSON_CONTAINS` pada array sederhana:

```
SELECT JSON_CONTAINS(
  JSON_ARRAY('buku', 'aksesori', 'perangkat'),
  JSON_QUOTE('buku')
) AS hasil;
```

Hasil `1` berarti nilai ditemukan.

Stokes juga membahas `JSON_CONTAINS()` sebagai fungsi yang mengembalikan `1` atau `0` untuk menunjukkan apakah candidate JSON ditemukan di target JSON.

6.10 Filter dengan JSON_OVERLAPS

`JSON_OVERLAPS` berguna untuk mengecek apakah dua dokumen JSON memiliki irisan nilai. Dokumentasi MySQL menjelaskan bahwa `JSON_OVERLAPS()` mengembalikan true jika dua dokumen memiliki pasangan key-value atau elemen array yang sama. (MySQL Developer Zone)

Contoh:

```
SELECT JSON_OVERLAPS(
  JSON_ARRAY('buku', 'aksesori'),
  JSON_ARRAY('perangkat', 'buku')
) AS ada_irisan;
```

Hasil:

```
1
```

Artinya ada elemen yang sama, yaitu `buku`.

Contoh praktis: filter order yang memiliki kategori yang masuk daftar minat laporan.

```
SELECT
  order_code,
  order_data->'$.items' AS items
FROM orders_json
WHERE JSON_OVERLAPS(
  JSON_EXTRACT(order_data, '$.items[*].kategori'),
  JSON_ARRAY('buku', 'aksesori')
);
```

Catatan kritis: query dengan wildcard pada JSON bisa berat untuk data besar. Untuk laporan serius, `JSON_TABLE` biasanya lebih jelas.

6.11 Filter dengan MEMBER OF

`MEMBER OF` digunakan untuk mengecek apakah sebuah nilai merupakan anggota array JSON. Dokumentasi MySQL menempatkan `MEMBER OF()` sebagai operator untuk pencarian nilai JSON. (MySQL Developer Zone)

Contoh sederhana:

```
SELECT 'buku' MEMBER OF (JSON_ARRAY('buku', 'aksesori',
'perangkat')) AS hasil;
```

Untuk angka:

```
SELECT 7 MEMBER OF (JSON_ARRAY(1, 3, 5, 7, 9)) AS hasil;
```

Stokes memberi contoh `MEMBER OF()` untuk mengecek nilai integer dalam array campuran dan menjelaskan bahwa hasil `1` berarti nilai ditemukan.

Contoh praktik pada tabel:

```
SELECT
  order_code,
  JSON_EXTRACT(order_data, '$.items[*].kategori') AS kategori
FROM orders_json
WHERE 'buku' MEMBER OF (JSON_EXTRACT(order_data,
'$.items[*].kategori'));
```

Jika query ini tidak berjalan sesuai versi server tertentu, gunakan `JSON_TABLE` sebagai pendekatan yang lebih stabil dan lebih mudah dibaca.

6.12 Mengubah JSON Menjadi Tabel dengan JSON_TABLE

`JSON_TABLE` adalah inti Bab 6. Fungsi ini mengubah array JSON menjadi bentuk tabel.

Dokumentasi MySQL menjelaskan bahwa `JSON_TABLE(expr, path COLUMNS (...))` mengekstrak data dari dokumen JSON dan mengembalikannya sebagai tabel relasional. (MySQL Developer Zone)

Contoh mengambil item order sebagai baris tabel:

```
SELECT
  o.order_code,
  jt.sku,
  jt.nama_produk,
  jt.kategori,
  jt.qty,
  jt.harga
FROM orders_json o,
JSON_TABLE(
  o.order_data,
  '$.items[*]'
  COLUMNS (
    sku VARCHAR(50) PATH '$.sku',
    nama_produk VARCHAR(100) PATH '$.nama_produk',
    kategori VARCHAR(50) PATH '$.kategori',
    qty INT PATH '$.qty',
    harga DECIMAL(12,2) PATH '$.harga'
  )
) AS jt;
```

Hasilnya kira-kira:

order_code	sku	nama_produk	kategori	qty	harga
ORD-2026-001	BK-JSON-001	Buku JSON Praktis	buku	2	75000
ORD-2026-001	USB-032	Flashdisk 32GB	aksesori	1	65000
ORD-2026-002	KB-MECH-001	Keyboard Mekanik	perangkat	1	350000

Inilah momen penting: array JSON berubah menjadi baris tabel.

Buku *Modern Oracle Database Programming* juga menekankan ide yang sama dalam konteks SQL/JSON: menyimpan JSON di database adalah satu hal, tetapi mengekstraknya menjadi format relasional adalah bagian penting dari bekerja dengan dokumen JSON.

6.13 Query Produk dalam Array Order

Cari semua order yang berisi SKU `BK-JSON-001`:

```
SELECT
  o.order_code,
  jt.sku,
  jt.nama_produk,
  jt.qty,
  jt.harga
FROM orders_json o,
JSON_TABLE(
  o.order_data,
  '$.items[*]'
  COLUMNS (
    sku VARCHAR(50) PATH '$.sku',
    nama_produk VARCHAR(100) PATH '$.nama_produk',
    qty INT PATH '$.qty',
    harga DECIMAL(12,2) PATH '$.harga'
  )
) AS jt
WHERE jt.sku = 'BK-JSON-001';
```

Cari semua item kategori buku:

```
SELECT
  o.order_code,
  jt.nama_produk,
  jt.kategori,
  jt.qty,
  jt.harga
FROM orders_json o,
JSON_TABLE(
  o.order_data,
  '$.items[*]'
  COLUMNS (
    nama_produk VARCHAR(100) PATH '$.nama_produk',
    kategori VARCHAR(50) PATH '$.kategori',
    qty INT PATH '$.qty',
    harga DECIMAL(12,2) PATH '$.harga'
  )
) AS jt
WHERE jt.kategori = 'buku';
```

6.14 Join Tabel Relasional dengan Data JSON

Sekarang gabungkan tabel `customers` dan hasil `JSON_TABLE`.

```
SELECT
  c.nama,
  c.kota,
  o.order_code,
  jt.nama_produk,
  jt.kategori,
  jt.qty,
  jt.harga,
  jt.qty * jt.harga AS subtotal
FROM customers c
JOIN orders_json o
  ON c.customer_id = o.customer_id
JOIN JSON_TABLE(
  o.order_data,
  '$.items[*]'
  COLUMNS (
    nama_produk VARCHAR(100) PATH '$.nama_produk',
    kategori VARCHAR(50) PATH '$.kategori',
    qty INT PATH '$.qty',
    harga DECIMAL(12,2) PATH '$.harga'
  )
) AS jt
ORDER BY c.kota, o.order_code;
```

Query ini menggabungkan:

- data customer dari tabel relasional;
- data order dari tabel JSON;
- item transaksi dari array JSON.

Inilah kekuatan desain hybrid.

Data yang stabil tetap relasional. Data yang fleksibel tetap JSON. Laporan tetap bisa dibuat.

6.15 Filter Customer Berdasarkan Kota

Filter transaksi customer dari Tangerang Selatan:

```
SELECT
  c.nama,
  c.kota,
  o.order_code,
  jt.nama_produk,
  jt.qty,
  jt.harga,
  jt.qty * jt.harga AS subtotal
FROM customers c
JOIN orders_json o
  ON c.customer_id = o.customer_id
JOIN JSON_TABLE(
  o.order_data,
  '$.items[*]'
  COLUMNS (
    nama_produk VARCHAR(100) PATH '$.nama_produk',
    qty INT PATH '$.qty',
    harga DECIMAL(12,2) PATH '$.harga'
  )
) AS jt
WHERE c.kota = 'Tangerang Selatan';
```

Catatan desain: karena `kota` berada di kolom relasional, filter menjadi sederhana dan jelas.

6.16 Membuat Laporan Total Order per Kota

```
SELECT
  c.kota,
  SUM(jt.qty * jt.harga) AS total_transaksi
FROM customers c
JOIN orders_json o
  ON c.customer_id = o.customer_id
JOIN JSON_TABLE(
  o.order_data,
  '$.items[*]'
  COLUMNS (
    qty INT PATH '$.qty',
    harga DECIMAL(12,2) PATH '$.harga'
  )
) AS jt
WHERE o.order_data->>'$.status' = 'paid'
GROUP BY c.kota
ORDER BY total_transaksi DESC;
```

Laporan ini menjawab:

- kota mana menghasilkan transaksi terbesar;
- hanya order dengan status `paid` yang dihitung;
- subtotal dihitung dari `qty * harga`.

6.17 Laporan Produk Terlaris

```
SELECT
  jt.sku,
  jt.nama_produk,
  SUM(jt.qty) AS total_qty,
  SUM(jt.qty * jt.harga) AS total_penjualan
FROM orders_json o
JOIN JSON_TABLE(
  o.order_data,
  '$.items[*]'
  COLUMNS (
    sku VARCHAR(50) PATH '$.sku',
    nama_produk VARCHAR(100) PATH '$.nama_produk',
    qty INT PATH '$.qty',
    harga DECIMAL(12,2) PATH '$.harga'
  )
) AS jt
WHERE o.order_data->>'$.status' = 'paid'
GROUP BY jt.sku, jt.nama_produk
ORDER BY total_qty DESC;
```

Makna strategis: JSON item order kini dapat menjadi laporan produk terlaris.

6.18 Aggregation dengan JSON_ARRAYAGG

`JSON_ARRAYAGG` mengubah hasil query menjadi array JSON. Dokumentasi MySQL menyatakan bahwa MySQL mendukung dua fungsi agregasi JSON, yaitu `JSON_ARRAYAGG()` dan `JSON_OBJECTAGG()`. (MySQL Developer Zone)

Contoh daftar order per kota dalam bentuk array:

```
SELECT
  c.kota,
  JSON_ARRAYAGG(o.order_code) AS daftar_order
FROM customers c
JOIN orders_json o
  ON c.customer_id = o.customer_id
GROUP BY c.kota;
```

Contoh membuat array nama produk per order:

```
SELECT
  o.order_code,
  JSON_ARRAYAGG(jt.nama_produk) AS produk_dibeli
FROM orders_json o
JOIN JSON_TABLE(
  o.order_data,
  '$.items[*]'
  COLUMNS (
    nama_produk VARCHAR(100) PATH '$.nama_produk'
  )
) AS jt
GROUP BY o.order_code;
```

Hasilnya cocok untuk format API atau laporan JSON.

6.19 Aggregation dengan JSON_OBJECTAGG

`JSON_OBJECTAGG` membuat object JSON dari pasangan key-value. Dokumentasi MySQL menjelaskan bahwa `JSON_OBJECTAGG()` mengembalikan result set sebagai object JSON. (MySQL Developer Zone)

Contoh total transaksi per kota sebagai object JSON:

```
SELECT
  JSON_OBJECTAGG(kota, total_transaksi) AS laporan_kota
FROM (
  SELECT
    c.kota,
    SUM(jt.qty * jt.harga) AS total_transaksi
  FROM customers c
  JOIN orders_json o
    ON c.customer_id = o.customer_id
  JOIN JSON_TABLE(
    o.order_data,
    '$.items[*]'
    COLUMNS (
      qty INT PATH '$.qty',
      harga DECIMAL(12,2) PATH '$.harga'
    )
  ) AS jt
  WHERE o.order_data->>'$.status' = 'paid'
  GROUP BY c.kota
) AS summary;
```

Contoh output:

```
{
  "Depok": 425000,
  "Tangerang Selatan": 425000
}
```

Kegunaan: hasil query langsung siap menjadi response API-style.

6.20 Membuat Laporan JSON Lengkap

Contoh membuat laporan order dalam format JSON per customer:

```
SELECT
  c.customer_id,
  c.nama,
  c.kota,
  JSON_ARRAYAGG(
    JSON_OBJECT(
      'order_code', o.order_code,
      'tanggal', o.order_data->>'$.tanggal',
      'status', o.order_data->>'$.status'
    )
  ) AS orders
FROM customers c
JOIN orders_json o
  ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.nama, c.kota;
```

Query ini mengubah data relasional menjadi JSON report.

Pola penting:

- `JSON_TABLE`: JSON → tabel.
- `JSON_ARRAYAGG`: tabel → array JSON.
- `JSON_OBJECTAGG`: tabel → object JSON.
- `JSON_OBJECT`: baris → object JSON.

6.21 Batasan Performa Query JSON

Query JSON sangat berguna, tetapi tidak gratis.

Risiko performa:

- membaca path JSON terlalu sering;
- memfilter array besar;
- memakai wildcard pada dokumen besar;
- tidak punya index;
- field penting tersembunyi dalam JSON;
- laporan berat langsung dari raw JSON;
- terlalu banyak nested object;
- struktur JSON tidak konsisten.

Stokes menekankan bahwa fungsi seperti `MEMBER OF`, `JSON_CONTAINS`, dan `JSON_OVERLAPS` dapat memanfaatkan *multi-valued index*, tetapi index juga punya batasan dan overhead saat insert/update.

Dokumentasi MySQL juga menyebut optimizer dapat memakai *multi-valued index* untuk `MEMBER OF()`, `JSON_CONTAINS()`, dan `JSON_OVERLAPS()` di klausa `WHERE`. (MySQL Developer Zone)

Pesan tajam: JSON cocok untuk fleksibilitas, tetapi laporan besar membutuhkan desain index dan strategi skema yang benar. Ini akan diperkuat pada Bab 7 tentang *indexing* dan *performance*.

Hands-on Bab 6

6.22 Membuat View Item Order

Agar query lebih rapi, buat view.

```
CREATE OR REPLACE VIEW order_items_view AS
SELECT
  o.order_id,
  o.order_code,
  o.customer_id,
  o.order_data->>'$.tanggal' AS tanggal,
  o.order_data->>'$.status' AS status,
  jt.sku,
  jt.nama_produk,
  jt.kategori,
  jt.qty,
  jt.harga,
  jt.qty * jt.harga AS subtotal
FROM orders_json o
JOIN JSON_TABLE(
  o.order_data,
  '$.items[*]'
  COLUMNS (
    sku VARCHAR(50) PATH '$.sku',
    nama_produk VARCHAR(100) PATH '$.nama_produk',
    kategori VARCHAR(50) PATH '$.kategori',
    qty INT PATH '$.qty',
    harga DECIMAL(12,2) PATH '$.harga'
  )
) AS jt;
```

Cek view:

```
SELECT * FROM order_items_view;
```

Manfaat view: mahasiswa tidak perlu menulis `JSON_TABLE` panjang berulang-ulang.

6.23 Laporan Total per Kota dengan View

```
SELECT
  c.kota,
  SUM(v.subtotal) AS total_transaksi
FROM customers c
JOIN order_items_view v
  ON c.customer_id = v.customer_id
WHERE v.status = 'paid'
GROUP BY c.kota
ORDER BY total_transaksi DESC;
```

6.24 Laporan Produk Terlaris dengan View

```
SELECT
  sku,
  nama_produk,
  SUM(qty) AS total_qty,
  SUM(subtotal) AS total_penjualan
FROM order_items_view
WHERE status = 'paid'
GROUP BY sku, nama_produk
ORDER BY total_qty DESC;
```

6.25 Export Laporan dengan Python

Buat file `export_sales_report.py`.

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

try:
    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor(dictionary=True)

    sql = """
```

```
            SELECT
              c.kota,
              SUM(v.subtotal) AS total_transaksi
            FROM customers c
            JOIN order_items_view v
              ON c.customer_id = v.customer_id
            WHERE v.status = 'paid'
            GROUP BY c.kota
            ORDER BY total_transaksi DESC
        """

        cursor.execute(sql)
        rows = cursor.fetchall()

        result = {
            "status": "success",
            "report": "total_transaksi_per_kota",
            "data": rows
        }

        with open("sales_report_by_city.json", "w",
    encoding="utf-8") as file:
            json.dump(result, file, ensure_ascii=False, indent=2,
    default=str)

        print("Export selesai: sales_report_by_city.json")

    except Error as err:
        print("Error database:", err)

    finally:
        if "cursor" in locals():
            cursor.close()
        if "conn" in locals() and conn.is_connected():
            conn.close()
```

Penjelasan singkat:

- `cursor(dictionary=True)`: hasil query mudah dijadikan JSON.
- `json.dump`: menulis output ke file.
- `default=str`: membantu menangani tipe angka/desimal tertentu saat export.
- laporan dibuat dari view `order_items_view`.

Mini-Lab Bab 6 — Laporan Transaksi dari Data JSON

Tujuan

Mahasiswa membuat laporan transaksi dari data order berbasis JSON. Fokusnya bukan hanya menyimpan order, tetapi mengubah isi JSON menjadi data analitik.\

Tugas Praktik

Kerjakan:

- Buat tabel `customers`.
- Buat tabel `orders_json`.
- Insert minimal 4 customer.
- Insert minimal 5 order dengan `items` berbentuk array JSON.
- Query status order dari JSON.
- Query metode pembayaran dari nested JSON.
- Filter order dengan status `paid`.
- Filter order yang mengandung produk tertentu.
- Ubah array `items` menjadi tabel memakai `JSON_TABLE`.
- Join `customers` dengan `orders_json`.
- Buat laporan total transaksi per kota.
- Buat laporan produk terlaris.
- Buat view `order_items_view`.
- Export laporan ke JSON memakai Python.

Output yang Diharapkan

Laporan total transaksi per kota:

```
{
  "status": "success",
  "report": "total_transaksi_per_kota",
  "data": [
    {
      "kota": "Tangerang Selatan",
      "total_transaksi": "425000.00"
    },
    {
      "kota": "Depok",
      "total_transaksi": "425000.00"
    }
  ]
}
```

Laporan produk terlaris:

```
BK-JSON-001 | Buku JSON Praktis | 3 | 225000
USB-032     | Flashdisk 32GB    | 3 | 195000
```

Kesalahan Umum Mahasiswa

Hindari kesalahan berikut:

- Menyimpan data transaksi sebagai JSON tetapi tidak bisa membuat laporan.
- Mengambil item array hanya dengan `$.items[0]`, padahal order bisa punya banyak item.
- Menggunakan query panjang berulang tanpa membuat view.
- Memakai JSON untuk semua field, termasuk field yang sering difilter.
- Tidak membedakan `JSON_CONTAINS`, `JSON_OVERLAPS`, dan `MEMBER OF`.
- Menganggap semua query JSON akan cepat.
- Tidak memakai `EXPLAIN` untuk query yang berat.
- Tidak menyiapkan strategi index untuk field penting.
- Membuat struktur JSON item tidak konsisten.
- Mengubah laporan langsung dari raw JSON tanpa validasi data.

Checklist Akhir Bab

Mahasiswa dinyatakan memahami Bab 6 jika mampu:

- Membaca nested JSON.
- Memfilter data berdasarkan field JSON.
- Menggunakan `JSON_CONTAINS`.
- Menggunakan `JSON_OVERLAPS`.
- Menggunakan `MEMBER OF`.
- Menggunakan `JSON_TABLE`.
- Mengubah array item order menjadi baris tabel.
- Join tabel customer dengan data JSON.
- Menghitung subtotal dari item JSON.
- Membuat laporan total transaksi per kota.
- Membuat laporan produk terlaris.
- Membuat agregasi JSON dengan `JSON_ARRAYAGG`.
- Membuat object laporan dengan `JSON_OBJECTAGG`.
- Export laporan ke JSON memakai Python.
- Menjelaskan batasan performa query JSON.

Ringkasan Bab

Bab ini membawa mahasiswa dari sekadar menyimpan JSON menuju kemampuan analisis. Mahasiswa belajar membaca nested JSON, memfilter object dan array, memakai `JSON_CONTAINS`, `JSON_OVERLAPS`, dan `MEMBER OF`, lalu mengubah array JSON menjadi tabel dengan `JSON_TABLE`. Setelah data JSON menjadi tabular, mahasiswa dapat melakukan join, agregasi, dan membuat laporan transaksi. **Pelajaran utama Bab 6: JSON yang baik bukan hanya fleksibel saat disimpan, tetapi juga kuat saat dianalisis dan dijadikan keputusan.**

Bab 7 — Indexing, Performance, dan Desain Skema Hybrid

> **JSON yang tidak didesain akan menjadi kuburan performa: fleksibel saat input, tetapi lambat saat dicari.**

Bab 7 diarahkan agar mahasiswa tidak hanya bisa membuat sistem, tetapi juga membuat sistem yang masuk akal secara performa. Strategi utamanya adalah **data penting menjadi kolom biasa**, **data fleksibel masuk JSON**, dan **data yang sering dicari wajib punya strategi index**.

Tujuan Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa mampu:

- Menjelaskan mengapa query JSON bisa lambat.
- Memahami bahwa kolom JSON **tidak diindeks langsung**.
- Membuat *generated column* dari field JSON penting.
- Membuat index pada *generated column*.
- Membandingkan query sebelum dan sesudah index.
- Menggunakan `EXPLAIN` untuk membaca rencana eksekusi query.
- Memakai *multi-valued index* untuk array JSON.
- Menilai kapan normalisasi lebih tepat daripada JSON.
- Memahami dampak index terhadap `INSERT` dan `UPDATE`.
- Mengoptimasi tabel profil mahasiswa berbasis JSON.

7.1 Mengapa Query JSON Bisa Lambat?

JSON memberi fleksibilitas. Tetapi fleksibilitas itu punya biaya.

Query JSON bisa lambat karena:

- nilai yang dicari berada di dalam dokumen JSON;
- database harus mengekstrak nilai dari path seperti `$.email`;
- field penting tidak punya index;
- array JSON perlu dibaca satu per satu;
- struktur JSON terlalu dalam;
- key tidak konsisten;
- tipe data tidak konsisten;
- query memakai fungsi seperti `JSON_EXTRACT` pada banyak baris;
- data besar tetapi desain index tidak ada.

Contoh query yang berpotensi lambat:

```
SELECT
  id,
  nama,
  profile->>'$.email' AS email
FROM mahasiswa_profile
WHERE profile->>'$.email' = 'alya@example.test';
```

Query ini terlihat sederhana. Tetapi jika tabel berisi ratusan ribu baris, database dapat membaca banyak baris untuk mengekstrak `$.email` satu per satu.

Buku Stokes menjelaskan bahwa MySQL menyimpan JSON dalam format internal yang dioptimalkan untuk pencarian berdasarkan key atau posisi array. Namun, format internal ini **bukan berarti semua pencarian otomatis cepat**. Untuk query yang sering dipakai, tetap diperlukan strategi index.

Pesan penting: JSON native lebih baik daripada teks mentah, tetapi tetap perlu desain performa.

7.2 Kolom JSON Tidak Diindeks Langsung

MySQL tidak mengindeks kolom JSON secara langsung seperti kolom `VARCHAR` atau `INT`.

Dokumentasi resmi MySQL menyatakan bahwa kolom `JSON`, seperti kolom tipe biner lain, tidak diindeks langsung. Strateginya adalah membuat index pada *generated column* yang mengekstrak nilai scalar dari kolom JSON. Dokumentasi yang sama juga menyebut InnoDB mendukung *multi-valued index* untuk array JSON. (MySQL Developer Zone)

Artinya, query seperti ini:

```
WHERE profile->>'$.email' = 'alya@example.test'
```

tidak otomatis cepat hanya karena kolom `profile` bertipe JSON.

Strategi yang benar:

- ekstrak `$.email` menjadi *generated column*;
- buat index pada *generated column* tersebut;
- gunakan kolom hasil ekstraksi untuk pencarian cepat.

7.3 Desain Skema Hybrid: Kunci Performa JSON

Skema hybrid adalah desain yang menggabungkan:

- kolom relasional untuk data penting;
- kolom JSON untuk data fleksibel;
- *generated column* untuk field JSON yang sering dicari;
- index untuk query yang sering dipakai.

Prinsip skema hybrid:

- **Data inti** menjadi kolom biasa.
- **Data fleksibel** masuk JSON.
- **Data yang sering dicari** harus punya strategi index.
- **Data yang sering dihitung** sebaiknya menjadi kolom biasa.
- **Data yang sering dipakai join** jangan disembunyikan di JSON.
- **Data array** yang sering dicari perlu *multi-valued index* atau desain tabel terpisah.

Contoh desain buruk:

```
CREATE TABLE users_buruk (
  id INT AUTO_INCREMENT PRIMARY KEY,
  data JSON NOT NULL
);
```

Masalah:

- email tersembunyi;
- nomor HP tersembunyi;
- kota tersembunyi;
- prodi tersembunyi;
- query laporan sulit;
- index sulit;
- debugging sulit.

Contoh desain lebih baik:

```
CREATE TABLE users_hybrid (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL,
  nama VARCHAR(100) NOT NULL,
  profile JSON NOT NULL
);
```

Lebih baik lagi jika field yang sering dicari diberi *generated column*.

7.4 Generated Column untuk Field JSON Penting

Generated column adalah kolom yang nilainya dihitung dari ekspresi lain.

Untuk JSON, kita dapat membuat kolom yang isinya berasal dari path tertentu.

Contoh:

```
email VARCHAR(150)
GENERATED ALWAYS AS (profile->>'$.email') STORED
```

Artinya:

- `email` tidak diinput manual;
- nilainya otomatis diambil dari `profile->>'$.email'`;
- jika `profile` berubah, nilai `email` ikut dihitung ulang;
- kolom ini bisa diberi index.

Dokumentasi MySQL menjelaskan bahwa JSON column dapat diindeks secara tidak langsung dengan membuat *generated column* yang mengekstrak informasi yang perlu diindeks, lalu membuat index pada kolom tersebut. (MySQL Developer Zone)

7.5 Virtual vs Stored Generated Column

Generated column umumnya memiliki dua pendekatan:

- **VIRTUAL**: dihitung saat dibaca.
- **STORED**: disimpan secara fisik setelah dihitung.

Untuk pembelajaran dan optimasi yang mudah dipahami, gunakan `STORED`.

Contoh:

```
email_gen VARCHAR(150)
GENERATED ALWAYS AS (profile->>'$.email') STORED
```

Kelebihan `STORED`:

- hasil ekstraksi disimpan;
- lebih mudah dipahami;
- lebih jelas saat diberi index;
- cocok untuk latihan performa.

Kekurangan:

- butuh ruang simpan tambahan;
- saat `INSERT` atau `UPDATE`, nilai perlu dihitung ulang.

Aturan praktis: jika field sering dicari, biaya simpan tambahan bisa layak.

7.6 Membuat Tabel Profil Mahasiswa untuk Optimasi

Buat tabel baru untuk latihan.

```
DROP TABLE IF EXISTS mahasiswa_profile_opt;

CREATE TABLE mahasiswa_profile_opt (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL,
  nama VARCHAR(100) NOT NULL,
  profile JSON NOT NULL,
  email_gen VARCHAR(150)
    GENERATED ALWAYS AS (profile->>'$.email') STORED,
  hp_gen VARCHAR(30)
    GENERATED ALWAYS AS (profile->>'$.kontak.hp') STORED,
  kota_gen VARCHAR(100)
    GENERATED ALWAYS AS (profile->>'$.alamat.kota') STORED,
  INDEX idx_email_gen (email_gen),
  INDEX idx_hp_gen (hp_gen),
  INDEX idx_kota_gen (kota_gen)
);
```

Penjelasan:

- `profile`: menyimpan data fleksibel.
- `email_gen`: mengambil `$.email`.
- `hp_gen`: mengambil `$.kontak.hp`.
- `kota_gen`: mengambil `$.alamat.kota`.
- `idx_email_gen`: index untuk pencarian email.
- `idx_hp_gen`: index untuk pencarian nomor HP.
- `idx_kota_gen`: index untuk pencarian kota.

Ini skema hybrid yang sehat: data fleksibel tetap JSON, tetapi field penting punya jalur cepat.

7.7 Insert Dataset Latihan

```
INSERT INTO mahasiswa_profile_opt (nim, nama, profile)
VALUES
(
  '2026001',
  'Alya Pratama',
  JSON_OBJECT(
    'email', 'alya@example.test',
    'kontak', JSON_OBJECT('hp', '081200000001'),
    'alamat', JSON_OBJECT('kota', 'Tangerang Selatan',
'provinsi', 'Banten'),
    'skill', JSON_ARRAY('python', 'sql', 'database')
  )
),
(
  '2026002',
  'Bima Santoso',
  JSON_OBJECT(
    'email', 'bima@example.test',
    'kontak', JSON_OBJECT('hp', '081200000002'),
    'alamat', JSON_OBJECT('kota', 'Depok', 'provinsi', 'Jawa
Barat'),
    'skill', JSON_ARRAY('python', 'api', 'automation')
  )
),
(
  '2026003',
  'Citra Lestari',
  JSON_OBJECT(
    'email', 'citra@example.test',
    'kontak', JSON_OBJECT('hp', '081200000003'),
    'alamat', JSON_OBJECT('kota', 'Bogor', 'provinsi', 'Jawa
Barat'),
    'skill', JSON_ARRAY('sql', 'data', 'database')
  )
);
```

Cek hasil:

```
SELECT
  nim,
  nama,
  email_gen,
  hp_gen,
  kota_gen,
  JSON_PRETTY(profile) AS profile_rapi
FROM mahasiswa_profile_opt;
```

7.8 Membandingkan Query Tanpa dan Dengan Generated Column

Query langsung dari JSON:

```
EXPLAIN
SELECT
  nim,
  nama
FROM mahasiswa_profile_opt
WHERE profile->>'$.email' = 'alya@example.test';
```

Query memakai *generated column*:

```
EXPLAIN
SELECT
  nim,
  nama
FROM mahasiswa_profile_opt
WHERE email_gen = 'alya@example.test';
```

Perhatikan kolom dalam hasil `EXPLAIN`:

- `type`
- `possible_keys`
- `key`
- `rows`
- `Extra`

Jika index digunakan, kolom `key` biasanya menampilkan nama index seperti `idx_email_gen`.

Dokumentasi MySQL menjelaskan bahwa optimizer dapat memakai index pada *generated column* dalam rencana eksekusi query, termasuk saat klausa `WHERE` menggunakan kolom tersebut. Dokumentasi itu juga menunjukkan penggunaan `EXPLAIN` untuk melihat apakah index digunakan. (MySQL Developer Zone)

7.9 Membaca Hasil EXPLAIN

`EXPLAIN` membantu mahasiswa memahami bagaimana database menjalankan query.

Kolom penting:

Kolom	Makna Praktis
`type`	cara akses tabel
`possible_keys`	index yang mungkin dipakai
`key`	index yang benar-benar dipakai
`rows`	perkiraan baris yang dibaca
`Extra`	informasi tambahan

Interpretasi sederhana:

- `type = ALL`: kemungkinan full table scan.
- `key = NULL`: tidak ada index yang dipakai.
- `key = idx_email_gen`: index dipakai.
- `rows` kecil: lebih efisien.
- `rows` besar: perlu diperiksa.

Buku Stokes memberi contoh `EXPLAIN` pada query JSON tanpa *multi-valued index* yang menunjukkan `type: ALL`, `key: NULL`, dan jumlah baris besar; setelah memakai pendekatan index yang tepat, query dapat memakai index.

Pesan tajam: jangan menebak performa. Gunakan `EXPLAIN`.

7.10 Studi Kasus: Mencari User Berdasarkan Nomor HP dalam JSON

Query lambat jika langsung dari JSON:

```
EXPLAIN
SELECT
  nim,
  nama
FROM mahasiswa_profile_opt
WHERE profile->>'$.kontak.hp' = '081200000002';
```

Query lebih sehat:

```
EXPLAIN
SELECT
  nim,
  nama
FROM mahasiswa_profile_opt
WHERE hp_gen = '081200000002';
```

Ambil datanya:

```
SELECT
  nim,
  nama,
  hp_gen AS nomor_hp
FROM mahasiswa_profile_opt
WHERE hp_gen = '081200000002';
```

Prinsip desain: jika nomor HP sering dipakai untuk pencarian, jangan hanya menyembunyikannya di JSON. Buat *generated column* atau jadikan kolom biasa.

7.11 Studi Kasus: Mencari Mahasiswa Berdasarkan Kota

Query langsung dari JSON:

```
SELECT
  nim,
  nama,
  profile->>'$.alamat.kota' AS kota
FROM mahasiswa_profile_opt
WHERE profile->>'$.alamat.kota' = 'Depok';
```

Query memakai *generated column*:

```
SELECT
  nim,
  nama,
  kota_gen AS kota
FROM mahasiswa_profile_opt
WHERE kota_gen = 'Depok';
```

Jika kota sering dipakai untuk laporan, bahkan lebih baik kota disimpan sebagai kolom biasa.

Contoh desain lebih tegas:

```
CREATE TABLE mahasiswa_profile_hybrid (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL,
  nama VARCHAR(100) NOT NULL,
  kota VARCHAR(100) NOT NULL,
  profile JSON NOT NULL,
  INDEX idx_kota (kota)
);
```

Pertanyaan desain: apakah `kota` data inti atau data fleksibel?
Jika sering dicari dan dilaporkan, jadikan kolom biasa.

7.12 Multi-Valued Index untuk Array JSON

Masalah berikutnya adalah array.

Contoh:

```
{
  "skill": ["python", "sql", "database"]
}
```

Jika kita sering mencari mahasiswa dengan skill tertentu, query array bisa berat.

MySQL InnoDB mendukung *multi-valued index* untuk array JSON. Dokumentasi `CREATE INDEX` menjelaskan bahwa `key_part` dapat memakai bentuk `(CAST json_expression AS type ARRAY)` untuk membuat *multi-valued index* pada kolom JSON. (MySQL Developer Zone)

Contoh tabel khusus latihan skill:

```
DROP TABLE IF EXISTS mahasiswa_skill_mvi;

CREATE TABLE mahasiswa_skill_mvi (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL,
  nama VARCHAR(100) NOT NULL,
  profile JSON NOT NULL,
  INDEX idx_skill_mvi (
    (CAST(profile->'$.skill' AS CHAR(50) ARRAY))
  )
);
```

Insert data:

```
INSERT INTO mahasiswa_skill_mvi (nim, nama, profile)
VALUES
(
  '2026001',
  'Alya Pratama',
  JSON_OBJECT('skill', JSON_ARRAY('python', 'sql',
'database'))
),
(
  '2026002',
  'Bima Santoso',
  JSON_OBJECT('skill', JSON_ARRAY('python', 'api',
'automation'))
),
(
  '2026003',
  'Citra Lestari',
  JSON_OBJECT('skill', JSON_ARRAY('sql', 'data', 'database'))
);
```

Cari mahasiswa dengan skill `python`:

```
EXPLAIN
SELECT
  nim,
  nama,
  profile->'$.skill' AS skill
FROM mahasiswa_skill_mvi
WHERE 'python' MEMBER OF (profile->'$.skill');
```

Stokes menjelaskan bahwa *multi-valued index* berguna untuk mencari nilai dalam array JSON, dan fungsi seperti `MEMBER OF`, `JSON_CONTAINS`, serta `JSON_OVERLAPS` dapat memanfaatkan MVI di klausa `WHERE`.

7.13 MEMBER OF, JSON_CONTAINS, dan JSON_OVERLAPS dengan MVI

Tiga pola pencarian array:

MEMBER OF

```
SELECT
  nim,
  nama
FROM mahasiswa_skill_mvi
WHERE 'python' MEMBER OF (profile->'$.skill');
```

Cocok untuk mencari satu nilai di array.

JSON_CONTAINS

```
SELECT
  nim,
  nama
FROM mahasiswa_skill_mvi
WHERE JSON_CONTAINS(profile->'$.skill',
JSON_QUOTE('database'));
```

Cocok untuk mengecek apakah array mengandung nilai tertentu.

JSON_OVERLAPS

```
SELECT
  nim,
  nama
FROM mahasiswa_skill_mvi
WHERE JSON_OVERLAPS(
  profile->'$.skill',
  JSON_ARRAY('python', 'security')
);
```

Cocok untuk mencari irisan antara dua array.

Dokumentasi MySQL menyebut optimizer dapat memakai *multi-valued index* untuk `MEMBER OF()`, `JSON_CONTAINS()`, dan `JSON_OVERLAPS()` di klausa `WHERE`. (MySQL Developer Zone)

7.14 Dampak Index terhadap Insert dan Update

Index mempercepat pencarian, tetapi memperlambat sebagian operasi tulis.

Saat ada index, database perlu:

- menyimpan data utama;
- menghitung nilai *generated column*;
- memperbarui index;
- memperbarui statistik;
- menjaga konsistensi struktur.

Untuk *multi-valued index*, satu baris dapat menghasilkan banyak entri index karena satu array punya banyak nilai.

Stokes mengingatkan bahwa DML pada MVI bekerja seperti index lain, tetapi satu baris insert/update/delete dapat menghasilkan lebih banyak entri index, sehingga overhead dapat memengaruhi performa. Ia juga mencatat bahwa array kosong tidak masuk index dan MVI tidak cocok untuk primary key.

Kesimpulan praktis:

- index mempercepat `SELECT`;
- index menambah beban `INSERT`;
- index menambah beban `UPDATE`;
- index butuh ruang simpan;
- index harus dibuat berdasarkan kebutuhan query nyata.

Jangan membuat index hanya karena bisa. Buat index karena query membutuhkannya.

7.15 Normalisasi vs Fleksibilitas

JSON bukan pengganti normalisasi.

Gunakan normalisasi jika:

- data sering di-*join*;
- data punya relasi kuat;
- data punya aturan konsisten;
- data sering dihitung;
- data sering dicari;
- data menjadi inti bisnis.

Gunakan JSON jika:

- data fleksibel;
- atribut berbeda antar record;
- data berasal dari API;
- metadata sering berubah;
- field jarang dipakai filter;
- struktur belum stabil.

Contoh `skill` mahasiswa.

Jika hanya dipakai sebagai profil fleksibel, boleh di JSON:

```
{
  "skill": ["python", "sql", "database"]
}
```

Jika skill sering dipakai untuk laporan, pencarian, rekomendasi, atau relasi ke kelas, lebih baik normalisasi:

```
CREATE TABLE mahasiswa_skill (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL,
  skill VARCHAR(50) NOT NULL,
  INDEX idx_skill (skill)
);
```

Prinsip strategis: jika data menjadi pusat analisis, jangan sembunyikan terlalu dalam di JSON.

Hands-on Bab 7

7.16 Membandingkan Query Sebelum dan Sesudah Index

Query langsung dari JSON

```
EXPLAIN
SELECT
  nim,
  nama
FROM mahasiswa_profile_opt
WHERE profile->>'$.email' = 'alya@example.test';
```

Perhatikan apakah:

- `type` masih `ALL`;
- `key` masih `NULL`;
- `rows` besar.

Query memakai generated column

```
EXPLAIN
SELECT
  nim,
  nama
FROM mahasiswa_profile_opt
WHERE email_gen = 'alya@example.test';
```

Perhatikan apakah:

- `key` memakai `idx_email_gen`;
- `rows` lebih kecil;
- query lebih jelas.

7.17 Membuat Index Setelah Tabel Ada

Jika tabel sudah dibuat tanpa index, tambahkan index.

```
CREATE INDEX idx_email_gen
ON mahasiswa_profile_opt (email_gen);
```

Dokumentasi MySQL menjelaskan bahwa `CREATE INDEX` digunakan untuk menambahkan index ke tabel yang sudah ada, dan pada dasarnya dipetakan ke operasi `ALTER TABLE`. (MySQL Developer Zone)

Cek index:

```
SHOW INDEX FROM mahasiswa_profile_opt;
```

7.18 Studi Kasus Produk Berdasarkan Tag dalam Array

Buat tabel produk:

```
DROP TABLE IF EXISTS produk_tag_mvi;

CREATE TABLE produk_tag_mvi (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nama_produk VARCHAR(100) NOT NULL,
  kategori VARCHAR(50) NOT NULL,
  spesifikasi JSON NOT NULL,
  INDEX idx_tag_mvi (
    (CAST(spesifikasi->'$.tag' AS CHAR(50) ARRAY))
  )
);
```

Insert data:

```
INSERT INTO produk_tag_mvi (nama_produk, kategori,
spesifikasi)
VALUES
(
  'Laptop Edukasi 14',
  'laptop',
  JSON_OBJECT(
    'tag', JSON_ARRAY('edukasi', 'portable', 'python'),
    'ram_gb', 8
  )
),
(
  'Router Lab Kampus',
  'jaringan',
  JSON_OBJECT(
    'tag', JSON_ARRAY('jaringan', 'lab', 'praktikum'),
    'jumlah_port', 4
  )
),
(
  'Sensor Suhu IoT',
  'sensor',
  JSON_OBJECT(
    'tag', JSON_ARRAY('iot', 'sensor', 'praktikum'),
    'tegangan_volt', 3.3
  )
);
```

Cari produk dengan tag `praktikum`:

```
EXPLAIN
SELECT
  nama_produk,
  kategori,
  spesifikasi->'$.tag' AS tag
FROM produk_tag_mvi
WHERE 'praktikum' MEMBER OF (spesifikasi->'$.tag');
```

Query hasil:

```
SELECT
  nama_produk,
  kategori,
  spesifikasi->'$.tag' AS tag
FROM produk_tag_mvi
WHERE 'praktikum' MEMBER OF (spesifikasi->'$.tag');
```

7.19 Export Hasil Optimasi dengan Python

Buat file `export_optimized_profiles.py`.

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

try:
    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor(dictionary=True)

    sql = """
        SELECT
          nim,
          nama,
          email_gen AS email,
          hp_gen AS nomor_hp,
          kota_gen AS kota
        FROM mahasiswa_profile_opt
        WHERE kota_gen = %s
        ORDER BY nama
    """

    cursor.execute(sql, ("Depok",))
    rows = cursor.fetchall()

    result = {
        "status": "success",
        "query": "mahasiswa_by_kota_indexed",
        "count": len(rows),
        "data": rows
    }

    with open("optimized_profiles.json", "w",
encoding="utf-8") as file:
        json.dump(result, file, ensure_ascii=False, indent=2)

    print("Export selesai: optimized_profiles.json")

except Error as err:
```

```
        print("Error database:", err)

    finally:
        if "cursor" in locals():
            cursor.close()
        if "conn" in locals() and conn.is_connected():
            conn.close()
```

Jalankan:

```
python export_optimized_profiles.py
```

Mini-Lab Bab 7 — Optimasi Tabel Profil Mahasiswa

Tujuan

Mahasiswa mengoptimasi tabel profil mahasiswa berbasis JSON agar pencarian berdasarkan email, nomor HP, kota, dan skill menjadi lebih masuk akal secara performa.

Tugas Praktik

Kerjakan:

- Buat tabel `mahasiswa_profile_opt`.
- Tambahkan kolom `profile JSON`.
- Tambahkan *generated column*:
  - `email_gen`
  - `hp_gen`
  - `kota_gen`
- Buat index:
  - `idx_email_gen`
  - `idx_hp_gen`
  - `idx_kota_gen`
- Insert minimal 10 data mahasiswa.
- Jalankan query langsung dari JSON.
- Jalankan query melalui *generated column*.
- Bandingkan hasil `EXPLAIN`.
- Buat tabel `mahasiswa_skill_mvi`.
- Buat *multi-valued index* untuk array `skill`.
- Cari mahasiswa dengan skill `python`.
- Cari mahasiswa dengan skill `database`.
- Tulis refleksi pendek: kapan data lebih baik di JSON, kapan lebih baik dinormalisasi.

Output yang Diharapkan

Contoh output `optimized_profiles.json`:

```
{
  "status": "success",
  "query": "mahasiswa_by_kota_indexed",
  "count": 1,
  "data": [
    {
      "nim": "2026002",
      "nama": "Bima Santoso",
      "email": "bima@example.test",
      "nomor_hp": "081200000002",
      "kota": "Depok"
    }
  ]
}
```

Kesalahan Umum Mahasiswa

Hindari:

- Mengira kolom JSON otomatis cepat dicari.
- Menggunakan `JSON_EXTRACT` pada jutaan baris tanpa index.
- Menyimpan field penting hanya di JSON.
- Tidak memakai `EXPLAIN`.
- Membuat terlalu banyak index tanpa kebutuhan.
- Membuat index pada field yang jarang dicari.
- Tidak menghitung dampak index terhadap `INSERT` dan `UPDATE`.
- Menggunakan array JSON untuk data yang seharusnya menjadi tabel relasi.
- Menganggap *multi-valued index* menyelesaikan semua masalah array.
- Tidak menjaga konsistensi key JSON.
- Membandingkan angka JSON sebagai string tanpa `CAST`.

Checklist Akhir Bab

Mahasiswa dinyatakan memahami Bab 7 jika mampu:

- Menjelaskan mengapa query JSON bisa lambat.
- Menjelaskan bahwa kolom JSON tidak diindeks langsung.
- Membuat *generated column* dari `$.email`.
- Membuat index pada *generated column*.
- Menggunakan `EXPLAIN`.
- Membaca `possible_keys`, `key`, `rows`, dan `type`.
- Membandingkan query sebelum dan sesudah index.
- Membuat *generated column* untuk nomor HP.
- Membuat *multi-valued index* untuk array JSON.
- Menggunakan `MEMBER OF` untuk array JSON.
- Menjelaskan dampak index terhadap operasi tulis.
- Menjelaskan kapan memakai JSON dan kapan normalisasi.
- Membuat mini-lab optimasi tabel profil mahasiswa.

Ringkasan Bab

Bab ini menegaskan bahwa JSON bukan hanya soal fleksibilitas, tetapi juga performa. MySQL tidak mengindeks kolom JSON secara langsung, sehingga field penting perlu diekstrak ke *generated column* lalu diberi index. Untuk array JSON, *multi-valued index* dapat membantu pencarian dengan `MEMBER OF`, `JSON_CONTAINS`, atau `JSON_OVERLAPS`, tetapi tetap memiliki biaya saat insert dan update. Mahasiswa juga belajar memakai `EXPLAIN` agar tidak menebak-nebak performa. **Pelajaran utama Bab 7: desain JSON yang baik bukan hanya mudah diisi, tetapi cepat dicari, jelas dianalisis, dan tetap disiplin secara skema.**

Bab 8 — Validasi JSON dengan JSON Schema Sebelum Masuk Database

> **JSON valid belum tentu data benar; validasi sintaks hanya memastikan bentuknya sah, sedangkan JSON Schema memastikan strukturnya layak dipercaya.**

Bab 8 diletakkan sebagai penguatan penting karena MySQL dapat memvalidasi format JSON, tetapi tidak otomatis memastikan struktur bisnisnya benar. Disini juga menegaskan penggunaan JSON Schema untuk tipe data, field wajib, pola email, batas nilai, aturan input, validasi sebelum insert ke MySQL, dan posisi JSON Schema dalam pipeline Python serta n8n.

Tujuan Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa mampu:

- Menjelaskan mengapa **JSON valid belum tentu data benar**.
- Memahami konsep dasar **JSON Schema**.
- Menggunakan keyword `type`, `properties`, `required`, `items`, dan `enum`.
- Memvalidasi **object** dan **array**.
- Memvalidasi email, tanggal, angka, dan string.
- Mengenal `$schema` dan *dialect* JSON Schema.
- Mengurangi ambiguitas struktur data.
- Memvalidasi JSON sebelum insert ke MySQL.
- Menyimpan hanya data yang lolos validasi.
- Menempatkan JSON Schema dalam pipeline Python dan n8n.

8.1 Mengapa JSON Valid Belum Tentu Benar?

MySQL dapat menolak JSON yang rusak secara sintaks. Misalnya, koma berlebih atau tanda kutip yang salah. Tipe data `JSON` di MySQL memang memberi validasi otomatis terhadap dokumen JSON; dokumen tidak valid akan menghasilkan error. (MySQL Developer Zone)

Namun, MySQL tidak otomatis tahu aturan bisnis kita.

Contoh JSON valid:

```
{
  "nim": "2026001",
  "nama": "Alya Pratama",
  "email": "bukan-email",
  "umur": "dua puluh",
  "prodi": "Teknik Tidur Siang"
}
```

JSON di atas valid secara format. Tetapi datanya bermasalah:

- `email` bukan alamat email yang benar.
- `umur` seharusnya angka.
- `prodi` mungkin tidak ada dalam daftar prodi yang sah.
- Tidak ada aturan apakah `nim` wajib.
- Tidak ada aturan panjang minimal `nama`.

> **Masalahnya bukan pada JSON. Masalahnya pada tidak adanya kontrak data.**

Buku Stokes menegaskan bahwa menyimpan JSON sebagai string lama tidak memberi validasi, tidak menjaga tipe data, tidak menjaga rentang nilai, tidak memastikan tag konsisten, dan menyulitkan pencarian. Native JSON memperbaiki validasi format, tetapi validasi bisnis tetap perlu aturan tambahan.

8.2 Konsep JSON Schema

JSON Schema adalah cara formal untuk mendeskripsikan struktur dan aturan dokumen JSON.

JSON Schema menjawab pertanyaan:

- Field apa yang wajib ada?
- Tipe datanya apa?
- Apakah nilainya string, angka, array, atau object?
- Apakah email harus berbentuk email?
- Apakah umur minimal tertentu?
- Apakah prodi hanya boleh dari daftar tertentu?
- Apakah array harus berisi object dengan struktur tertentu?
- Apakah field tambahan boleh diterima?

JSON Schema secara resmi menggunakan keyword sebagai blok pembangun untuk mendefinisikan struktur dokumen JSON. (json-schema.org)

Itelman & Viotti menjelaskan JSON Schema sebagai bahasa deklaratif untuk mendefinisikan struktur dan makna dokumen JSON, atau *instance*. Mereka juga menegaskan bahwa schema adalah definisi formal struktur data yang membawa informasi tentang sintaks dan semantik data.

Makna praktis: JSON Schema adalah kontrak antara pengirim data, penerima data, aplikasi, database, dan workflow.

8.3 JSON Schema sebagai Pengurang Ambiguitas

Data rusak sering bukan karena teknologinya buruk. Data rusak sering terjadi karena tim tidak sepakat tentang arti field.

Contoh ambigu:

```
{
  "tanggal": "05/06/26"
}
```

Pertanyaannya:

- Apakah 5 Juni 2026?
- Apakah 6 Mei 2026?
- Apakah tahun 1926 atau 2026?
- Apakah formatnya selalu sama?

JSON Schema memaksa kita membuat aturan.

Contoh lebih jelas:

```
{
  "tanggal_daftar": "2026-06-05"
}
```

Dengan schema:

```
{
  "type": "string",
  "format": "date"
}
```

Itelman & Viotti menekankan bahwa schema mengurangi error dan meningkatkan komunikasi karena data yang dibagikan dapat divalidasi terhadap format yang diharapkan.

Pesan kritis: schema bukan sekadar teknis; schema adalah disiplin komunikasi data.

8.4 Mengenal $schema dan Dialect

Bagian `$schema` memberi tahu validator versi aturan JSON Schema yang digunakan.

Contoh:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Draft 2020-12 adalah salah satu *dialect* JSON Schema modern. Itelman & Viotti memakai contoh `$schema` Draft 2020-12 untuk menunjukkan versi schema yang digunakan dalam validasi data.

Namun, perlu hati-hati: MySQL memiliki fungsi `JSON_SCHEMA_VALID()` dan `JSON_SCHEMA_VALIDATION_REPORT()`, tetapi dokumentasi MySQL menyebut dukungan validasi schema mengikuti JSON Schema Draft 4. (MySQL Developer Zone)

Implikasi praktis:

- Untuk pipeline Python modern, kita dapat memakai Draft 2020-12.
- Untuk validasi langsung di MySQL, perhatikan kompatibilitas Draft 4.
- Jangan mencampur keyword modern tanpa mengecek dukungan validator.

Dalam buku ini, validasi utama dilakukan di **Python sebelum insert ke MySQL**, agar kontrol lebih jelas dan mudah diuji.

8.5 Keyword Penting JSON Schema

Keyword dasar yang wajib dikuasai mahasiswa:

Keyword	Fungsi
`$schema`	Menentukan versi/dialect schema
`type`	Menentukan tipe data
`properties`	Mendefinisikan field object
`required`	Menentukan field wajib
`items`	Mendefinisikan isi array
`enum`	Membatasi pilihan nilai
`minimum`	Batas angka minimum
`maximum`	Batas angka maksimum
`minLength`	Panjang minimum string
`maxLength`	Panjang maksimum string
`format`	Format logis seperti email atau date
`additionalProperties`	Mengatur apakah field tambahan boleh diterima

Itelman & Viotti menekankan filosofi constraint: setiap keyword assertion menambahkan batasan baru pada data yang dianggap valid.

Prinsip: semakin jelas schema, semakin kecil ruang data berantakan.

8.6 Validasi Object

Contoh data mahasiswa:

```
{
  "nim": "2026001",
  "nama": "Alya Pratama",
  "email": "alya@example.test",
  "prodi": "Informatika",
  "angkatan": 2026
}
```

Schema dasar:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Mahasiswa",
  "type": "object",
  "required": ["nim", "nama", "email", "prodi", "angkatan"],
  "properties": {
    "nim": {
      "type": "string",
      "minLength": 5
    },
    "nama": {
      "type": "string",
      "minLength": 3
    },
    "email": {
      "type": "string",
      "format": "email"
    },
    "prodi": {
      "type": "string",
      "enum": ["Informatika", "Sistem Informasi", "Teknik
Informatika"]
    },
    "angkatan": {
      "type": "integer",
      "minimum": 2020,
      "maximum": 2030
    }
  },
  "additionalProperties": false
}
```

Makna schema:

- root harus object;
- `nim`, `nama`, `email`, `prodi`, dan `angkatan` wajib ada;
- `email` harus string dengan format email;
- `prodi` hanya boleh dari daftar `enum`;
- `angkatan` harus integer antara 2020 dan 2030;
- field tambahan ditolak.

Catatan: `additionalProperties: false` membuat schema lebih ketat. Ini baik untuk data pendaftaran agar field liar tidak masuk.

8.7 Validasi Array

Jika file `students.json` berisi banyak mahasiswa, root-nya adalah array.

Contoh:

```
[
  {
    "nim": "2026001",
    "nama": "Alya Pratama",
    "email": "alya@example.test",
    "prodi": "Informatika",
    "angkatan": 2026
  }
]
```

Schema array:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Daftar Mahasiswa",
  "type": "array",
  "minItems": 1,
  "items": {
    "type": "object",
    "required": ["nim", "nama", "email", "prodi",
"angkatan"],
    "properties": {
      "nim": {
        "type": "string",
        "minLength": 5
      },
      "nama": {
        "type": "string",
```

```
        "minLength": 3
      },
      "email": {
        "type": "string",
        "format": "email"
      },
      "prodi": {
        "type": "string",
        "enum": ["Informatika", "Sistem Informasi", "Teknik
Informatika"]
      },
      "angkatan": {
        "type": "integer",
        "minimum": 2020,
        "maximum": 2030
      }
    },
    "additionalProperties": false
  }
}
```

Itelman & Viotti memberi contoh dataset schema yang merepresentasikan array of entries, dengan `type: "array"` dan `items` yang mengacu pada schema setiap entry.

8.8 Validasi Email, Tanggal, Angka, dan String

JSON Schema dapat memvalidasi bentuk data umum.

Contoh email:

```
{
  "type": "string",
  "format": "email"
}
```

Contoh tanggal:

```
{
  "type": "string",
  "format": "date"
}
```

Contoh angka:

```
{
  "type": "integer",
  "minimum": 2020,
  "maximum": 2030
}
```

Contoh string:

```
{
  "type": "string",
  "minLength": 3,
  "maxLength": 100
}
```

Catatan penting: pada Python `jsonschema`, validasi format seperti email atau date perlu `FormatChecker`. Dokumentasi `jsonschema` menunjukkan fungsi `validate()` sebagai cara sederhana untuk memvalidasi instance terhadap schema, dan error `ValidationError` akan muncul jika data melanggar aturan. (jsonschema)

8.9 Posisi JSON Schema dalam Pipeline

Pipeline sehat:

```
Input JSON
  ↓
Validasi JSON syntax
  ↓
Validasi JSON Schema
  ↓
Cleaning ringan
  ↓
Insert ke MySQL
  ↓
Query / reporting
  ↓
Export / workflow
```

Posisi JSON Schema:

- sebelum insert ke MySQL;
- sebelum workflow menyimpan data;
- sebelum API menerima payload;
- sebelum batch import;
- sebelum data dipakai laporan;
- sebelum data dikirim ke proses analitik.

Dalam konteks n8n:

```
Webhook → Code node validasi → MySQL node → Response
```

Dalam konteks Python:

```
students.json → jsonschema → mysql-connector-python → MySQL
```

Aturan utama: database bukan tempat sampah. Validasi harus dilakukan sebelum data masuk.

Hands-on Bab 8

8.10 Instalasi Pustaka Python

Aktifkan *virtual environment*:

```
source venv/bin/activate
```

Install pustaka:

```
pip install jsonschema mysql-connector-python
```

Cek:

```
python -c "import jsonschema; print('jsonschema siap')"
```

8.11 Membuat student_schema.json

Buat file `student_schema.json`.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Pendaftaran Mahasiswa",
  "type": "object",
  "required": ["nim", "nama", "email", "prodi", "angkatan",
"profile"],
  "properties": {
    "nim": {
      "type": "string",
      "minLength": 5,
      "maxLength": 20
    },
    "nama": {
      "type": "string",
      "minLength": 3,
      "maxLength": 100
    },
    "email": {
      "type": "string",
      "format": "email"
    },
    "prodi": {
      "type": "string",
      "enum": ["Informatika", "Sistem Informasi", "Teknik
Informatika"]
```

```
    },
    "angkatan": {
      "type": "integer",
      "minimum": 2020,
      "maximum": 2030
    },
    "tanggal_daftar": {
      "type": "string",
      "format": "date"
    },
    "profile": {
      "type": "object",
      "required": ["alamat", "skill"],
      "properties": {
        "alamat": {
          "type": "object",
          "required": ["kota", "provinsi"],
          "properties": {
            "kota": {
              "type": "string",
              "minLength": 3
            },
            "provinsi": {
              "type": "string",
              "minLength": 3
            }
          },
          "additionalProperties": false
        },
        "skill": {
          "type": "array",
          "minItems": 1,
          "items": {
            "type": "string",
            "minLength": 2
          }
        }
      },
      "additionalProperties": false
    }
  },
  "additionalProperties": false
}
```

Schema ini sengaja ketat agar data pendaftaran mahasiswa tidak liar.

8.12 Membuat students.json

Buat file `students.json`.

```
[
  {
    "nim": "2026001",
    "nama": "Alya Pratama",
    "email": "alya@example.test",
    "prodi": "Informatika",
    "angkatan": 2026,
    "tanggal_daftar": "2026-05-26",
    "profile": {
      "alamat": {
        "kota": "Tangerang Selatan",
        "provinsi": "Banten"
      },
      "skill": ["Python", "SQL", "Database"]
    }
  },
  {
    "nim": "2026002",
    "nama": "Bima Santoso",
    "email": "bima@example.test",
    "prodi": "Sistem Informasi",
    "angkatan": 2026,
    "tanggal_daftar": "2026-05-26",
    "profile": {
      "alamat": {
        "kota": "Depok",
        "provinsi": "Jawa Barat"
      },
      "skill": ["Python", "API"]
    }
  }
]
```

8.13 Validasi File students.json dengan Python

Buat file `validate_students.py`.

```
import json
from jsonschema import Draft202012Validator, FormatChecker


def load_json(filename):
    with open(filename, "r", encoding="utf-8") as file:
        return json.load(file)


schema = load_json("student_schema.json")
students = load_json("students.json")

validator = Draft202012Validator(
    schema,
    format_checker=FormatChecker()
)

errors_found = False

for index, student in enumerate(students):
    errors = sorted(
        validator.iter_errors(student),
        key=lambda error: list(error.path)
    )

    if errors:
        errors_found = True
        print(f"Data ke-{index + 1} tidak valid")

        for error in errors:
            lokasi = ".".join(str(item) for item in
error.path)
            print(f"- Lokasi: {lokasi if lokasi else
'root'}")
            print(f"  Pesan : {error.message}")
    else:
        print(f"Data ke-{index + 1} valid")

if errors_found:
    print("Validasi gagal. Data tidak boleh masuk database.")
else:
    print("Semua data valid. Data siap masuk database.")
```

Penjelasan:

- `Draft202012Validator`: validator untuk Draft 2020-12.
- `FormatChecker`: mengaktifkan pengecekan format seperti email dan date.
- `iter_errors`: mengumpulkan semua error, bukan berhenti pada error pertama.
- `error.path`: menunjukkan lokasi field bermasalah.
- `error.message`: pesan kesalahan.

8.14 Menolak Field Wajib yang Kosong

Ubah salah satu data:

```
{
  "nim": "2026003",
  "email": "citra@example.test",
  "prodi": "Teknik Informatika",
  "angkatan": 2026,
  "profile": {
    "alamat": {
      "kota": "Bogor",
      "provinsi": "Jawa Barat"
    },
    "skill": ["SQL"]
  }
}
```

Field `nama` hilang.

Saat script dijalankan, validator akan menolak data karena `nama` masuk daftar `required`.

Pesan: field wajib tidak boleh bergantung pada niat baik penginput data. Harus dipaksa oleh schema.

8.15 Menolak Email Invalid

Contoh data salah:

```
{
  "nim": "2026004",
  "nama": "Deni Prakoso",
  "email": "bukan-email",
  "prodi": "Informatika",
  "angkatan": 2026,
  "tanggal_daftar": "2026-05-26",
  "profile": {
    "alamat": {
      "kota": "Bekasi",
      "provinsi": "Jawa Barat"
    },
    "skill": ["Python"]
  }
}
```

Karena schema memakai:

```
"format": "email"
```

dan Python memakai `FormatChecker`, data ini akan ditolak.

8.16 Menolak Tipe Data yang Salah

Contoh data salah:

```
{
  "nim": "2026005",
  "nama": "Eka Lestari",
  "email": "eka@example.test",
  "prodi": "Sistem Informasi",
  "angkatan": "dua ribu dua puluh enam",
  "tanggal_daftar": "2026-05-26",
  "profile": {
    "alamat": {
      "kota": "Serpong",
      "provinsi": "Banten"
    },
    "skill": ["Data"]
  }
}
```

Field `angkatan` salah karena schema meminta `integer`.

```
"angkatan": {
  "type": "integer",
  "minimum": 2020,
  "maximum": 2030
}
```

Pesan kritis: angka yang disimpan sebagai string akan menyulitkan query, filter, dan laporan.

8.17 Simpan Hanya JSON yang Lolos Validasi ke MySQL

Buat tabel:

```
DROP TABLE IF EXISTS validated_students;

CREATE TABLE validated_students (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL UNIQUE,
  nama VARCHAR(100) NOT NULL,
  email VARCHAR(150) NOT NULL UNIQUE,
  prodi VARCHAR(100) NOT NULL,
  angkatan INT NOT NULL,
  profile JSON NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Buat file `import_valid_students.py`.

```
import json
import mysql.connector
from mysql.connector import Error
from jsonschema import Draft202012Validator, FormatChecker


DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}


def load_json(filename):
    with open(filename, "r", encoding="utf-8") as file:
        return json.load(file)


def validate_students(students, schema):
    validator = Draft202012Validator(
        schema,
        format_checker=FormatChecker()
    )

    valid_data = []
    invalid_data = []
```

```
    for index, student in enumerate(students):
        errors = sorted(
            validator.iter_errors(student),
            key=lambda error: list(error.path)
        )

        if errors:
            invalid_data.append({
                "row": index + 1,
                "nim": student.get("nim"),
                "errors": [
                    {
                        "path": ".".join(str(item) for item
in error.path),
                        "message": error.message
                    }
                    for error in errors
                ]
            })
        else:
            valid_data.append(student)

    return valid_data, invalid_data


def insert_students(valid_data):
    conn = None
    cursor = None

    try:
        conn = mysql.connector.connect(**DB_CONFIG)
        cursor = conn.cursor(prepared=True)
        conn.start_transaction()

        sql = """
            INSERT INTO validated_students
              (nim, nama, email, prodi, angkatan, profile)
            VALUES
              (%s, %s, %s, %s, %s, %s)
        """

        for student in valid_data:
            cursor.execute(sql, (
                student["nim"],
                student["nama"],
                student["email"],
                student["prodi"],
                student["angkatan"],
```

```
                json.dumps(student["profile"],
ensure_ascii=False)
            ))

        conn.commit()
        print(f"Insert berhasil: {len(valid_data)} data")

    except Error as err:
        if conn:
            conn.rollback()
        print("Error database:", err)

    finally:
        if cursor:
            cursor.close()
        if conn and conn.is_connected():
            conn.close()


def save_invalid_report(invalid_data):
    with open("invalid_students_report.json", "w",
encoding="utf-8") as file:
        json.dump(invalid_data, file, ensure_ascii=False,
indent=2)


if __name__ == "__main__":
    schema = load_json("student_schema.json")
    students = load_json("students.json")

    valid_data, invalid_data = validate_students(students,
schema)

    if invalid_data:
        save_invalid_report(invalid_data)
        print("Ada data invalid. Lihat
invalid_students_report.json")

    if valid_data:
        insert_students(valid_data)
    else:
        print("Tidak ada data valid untuk disimpan.")
```

Prinsip penting: data invalid tidak dibuang diam-diam. Simpan laporan error agar bisa diperbaiki.

8.18 Laporan Error yang Bisa Dibaca

Contoh `invalid_students_report.json`:

```
[
  {
    "row": 3,
    "nim": "2026004",
    "errors": [
      {
        "path": "email",
        "message": "'bukan-email' is not a 'email'"
      },
      {
        "path": "angkatan",
        "message": "'dua ribu dua puluh enam' is not of type
'integer'"
      }
    ]
  }
]
```

Laporan seperti ini penting untuk:

- debugging;
- dokumentasi kualitas data;
- audit sederhana;
- memperbaiki file input;
- komunikasi dengan pengirim data.

8.19 Validasi Tambahan di MySQL

Walaupun validasi utama dilakukan di Python, MySQL juga memiliki fungsi validasi schema.

Contoh konsep:

```
SELECT JSON_SCHEMA_VALID(@schema, @document);
```

MySQL mendukung `JSON_SCHEMA_VALID()` dan `JSON_SCHEMA_VALIDATION_REPORT()` untuk memvalidasi dokumen JSON terhadap schema Draft 4. (MySQL Developer Zone)

Stokes menunjukkan bahwa `JSON_SCHEMA_VALID()` mengembalikan `1` jika valid dan `0` jika gagal. Untuk mengetahui alasan gagal, `JSON_SCHEMA_VALIDATION_REPORT()` memberi laporan dalam format JSON, termasuk lokasi dokumen dan keyword schema yang gagal.

Sikap praktis buku ini:

- Python dipakai sebagai gerbang validasi utama.
- MySQL tetap menjadi lapisan pertahanan tambahan.
- Jangan hanya bergantung pada satu lapisan validasi.

8.20 Posisi JSON Schema dalam n8n

Dalam n8n, JSON Schema dapat dipakai sebagai gerbang sebelum data masuk MySQL.

Alur sederhana:

```
Webhook
  ↓
Code node: validasi JSON Schema
  ↓
IF node: valid / invalid
  ↓
MySQL node: insert hanya jika valid
  ↓
Response: sukses atau daftar error
```

Pembagian tanggung jawab:

- **Webhook** menerima data JSON.
- **Code node** menjalankan validasi.
- **IF node** memisahkan data valid dan invalid.
- **MySQL node** menyimpan data valid.
- **Response** memberi pesan jelas ke pengirim.

Disini juga ditegaskan bahwa n8n dipakai untuk alur menerima JSON, validasi, membersihkan data, dan menyimpan ke MySQL.

Catatan: untuk aturan validasi kompleks, lebih baik validasi dikerjakan di Python service atau script yang bisa diuji, lalu n8n menjadi orkestrator workflow.

Mini-Lab Bab 8 — Gerbang Validasi Data Pendaftaran Mahasiswa

Tujuan

Mahasiswa membuat pipeline kecil yang hanya menyimpan data pendaftaran mahasiswa jika lolos JSON Schema.

Struktur Folder

```
json-schema-gate/
├── student_schema.json
├── students.json
├── validate_students.py
├── import_valid_students.py
├── invalid_students_report.json
└── valid_students_result.json
```

Tugas Praktik

Kerjakan:

- Buat `student_schema.json`.
- Buat `students.json` berisi minimal 5 data.
- Masukkan 3 data valid.
- Masukkan 2 data invalid:
  - satu email salah;
  - satu field wajib hilang;
  - satu tipe data salah.
- Jalankan validasi dengan Python.
- Simpan data valid ke tabel `validated_students`.
- Simpan laporan data invalid ke `invalid_students_report.json`.
- Query ulang data dari MySQL.
- Export data valid menjadi `valid_students_result.json`.

Query Cek Data Valid

```
SELECT
  nim,
  nama,
  email,
  prodi,
  angkatan,
  JSON_PRETTY(profile) AS profile
FROM validated_students
ORDER BY nim;
```

Script Export Data Valid

Buat file `export_valid_students.py`.

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

try:
    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor(dictionary=True)

    sql = """
        SELECT
          nim,
          nama,
          email,
          prodi,
          angkatan,
          profile
        FROM validated_students
        ORDER BY nim
    """

    cursor.execute(sql)
    rows = cursor.fetchall()
```

```
        result = []

        for row in rows:
            result.append({
                "nim": row["nim"],
                "nama": row["nama"],
                "email": row["email"],
                "prodi": row["prodi"],
                "angkatan": row["angkatan"],
                "profile": json.loads(row["profile"])
            })

        with open("valid_students_result.json", "w",
    encoding="utf-8") as file:
            json.dump(result, file, ensure_ascii=False, indent=2)

        print("Export selesai: valid_students_result.json")

    except Error as err:
        print("Error database:", err)

    finally:
        if "cursor" in locals():
            cursor.close()
        if "conn" in locals() and conn.is_connected():
            conn.close()
```

Kesalahan Umum Mahasiswa

Hindari:

- Menganggap JSON valid berarti data sudah benar.
- Tidak memakai `required`.
- Tidak membatasi `enum` untuk field pilihan.
- Tidak memakai `FormatChecker` untuk email dan tanggal.
- Membiarkan field tambahan masuk tanpa kontrol.
- Memasukkan semua data ke database lalu baru memeriksa.
- Tidak menyimpan laporan error validasi.
- Membuat schema terlalu longgar.
- Membuat schema terlalu rumit sejak awal.
- Tidak menyamakan schema dengan kebutuhan database.
- Menggunakan schema modern tanpa mengecek dukungan validator yang dipakai.

Checklist Akhir Bab

Mahasiswa dinyatakan memahami Bab 8 jika mampu:

- Menjelaskan perbedaan JSON valid dan data benar.
- Membuat JSON Schema sederhana.
- Menggunakan `$schema`.
- Menggunakan `type`.
- Menggunakan `properties`.
- Menggunakan `required`.
- Menggunakan `items`.
- Menggunakan `enum`.
- Memvalidasi object.
- Memvalidasi array.
- Memvalidasi email dengan `FormatChecker`.
- Memvalidasi tanggal.
- Menolak field wajib yang hilang.
- Menolak tipe data yang salah.
- Menyimpan hanya data valid ke MySQL.
- Membuat laporan data invalid.
- Menjelaskan posisi JSON Schema dalam pipeline Python dan n8n.

Ringkasan Bab

Bab ini menegaskan bahwa validasi format saja tidak cukup. MySQL dapat menolak JSON yang rusak secara sintaks, tetapi struktur bisnis tetap harus dijaga dengan JSON Schema. Mahasiswa belajar membuat schema untuk data mahasiswa, menentukan field wajib, membatasi tipe data, memvalidasi email dan tanggal, menolak data invalid, serta menyimpan hanya data yang lolos validasi ke MySQL. JSON Schema ditempatkan sebagai **gerbang kualitas data** sebelum database, baik dalam pipeline Python maupun workflow n8n. **Pelajaran utama Bab 8: database yang baik tidak dimulai dari query, tetapi dari data yang sejak awal sudah divalidasi, konsisten, dan dapat dipercaya.**

Bab 9 — n8n untuk Automasi JSON dan MySQL

> **n8n bukan pengganti coding; n8n adalah penguat integrasi agar JSON, API, workflow, dan database bisa bergerak cepat tanpa kehilangan kontrol teknis.**

Bab 9 memperkenalkan n8n untuk automasi JSON dan MySQL, dengan fokus pada *webhook*, struktur data item/field/JSON, node penting seperti Webhook, Set, Code, HTTP Request, MySQL, serta alur menerima JSON → validasi → membersihkan data → simpan ke MySQL.

Tujuan Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa mampu:

- Menjelaskan konsep **workflow automation**.
- Memahami struktur data n8n: **item**, **field**, dan **JSON**.
- Membuat **Webhook** penerima JSON.
- Mengirim data ke webhook memakai `curl`.
- Mengirim data memakai **Hoppscotch** sebagai API client open source.
- Membersihkan field dengan node **Edit Fields / Set**.
- Menggunakan **Code node** untuk validasi sederhana.
- Menyimpan data JSON ke MySQL.
- Query MySQL dari n8n.
- Mengirim response JSON.
- Export dan import workflow sebagai file JSON.
- Menentukan kapan logic dikerjakan di n8n dan kapan dikerjakan di Python.

9.1 Mengapa n8n Penting untuk Mahasiswa?

Mahasiswa sering belajar database, Python, dan API sebagai bagian terpisah. Padahal di dunia kerja, sistem berjalan sebagai alur.

Contoh alur nyata:

```
Form pendaftaran
  ↓
Webhook menerima JSON
  ↓
Data dibersihkan
  ↓
Data divalidasi
  ↓
Data disimpan ke MySQL
  ↓
Response dikirim ke client
```

n8n membantu membuat alur seperti ini lebih cepat. Setiap langkah dibuat sebagai node. Node disambungkan menjadi workflow.

n8n berguna untuk:

- prototyping API sederhana;
- menerima payload JSON;
- menghubungkan API dan database;
- membuat otomasi import data;
- menguji integrasi sebelum dibuat aplikasi penuh;
- membuat workflow yang bisa dibaca visual;
- mengurangi pekerjaan scripting berulang.

Pesan kritis: n8n mempercepat integrasi, tetapi logic penting tetap harus jelas, teruji, dan terdokumentasi.

9.2 Konsep Workflow Automation

Workflow automation adalah cara menyusun langkah kerja otomatis.

Dalam n8n:

- workflow adalah rangkaian node;
- node adalah satu langkah kerja;
- data bergerak dari satu node ke node berikutnya;
- setiap node menerima input dan menghasilkan output;
- output biasanya berbentuk JSON;
- workflow dapat dimulai oleh trigger seperti webhook.

Dokumentasi n8n menjelaskan bahwa Webhook node adalah trigger node yang dapat memulai workflow ketika menerima data dari aplikasi atau layanan lain, dan dapat dipakai untuk memproses data serta mengembalikan hasil seperti endpoint API. (n8n Docs)

Makna praktis: n8n dapat menjadi jembatan ringan antara client, JSON, Python logic, dan MySQL.

9.3 Struktur Data n8n: Item, Field, dan JSON

n8n mengirim data antar node dalam bentuk array object JSON. Elemen dalam array itu disebut **item**. Dokumentasi n8n menjelaskan bahwa data dari satu node ke node lain dikirim sebagai array JSON object, dan setiap elemen disebut item. (n8n Docs)

Struktur konsep:

```
[
  {
    "json": {
      "nim": "2026001",
      "nama": "Alya Pratama",
      "email": "alya@example.test"
    }
  }
]
```

Makna:

- array `[]` berisi daftar item;
- setiap item punya key `json`;
- field data berada di dalam `json`;
- node berikutnya membaca data dari item tersebut.

Dalam n8n versi baru, Code node dapat otomatis menambahkan key `json` jika belum ada dan membungkus item dalam array jika diperlukan. (n8n Docs)

Prinsip penting: pahami bentuk data sebelum membuat workflow. Banyak error n8n terjadi bukan karena node salah, tetapi karena struktur JSON tidak dipahami.

9.4 Node Penting dalam Bab Ini

Bab ini memakai node dasar.

Webhook

Fungsi:

- menerima request dari luar;
- menjadi pintu masuk JSON;
- menjalankan workflow saat dipanggil;
- dapat mengembalikan response.

Edit Fields / Set

Node ini digunakan untuk menata field. Dokumentasi n8n menyebut Edit Fields node dapat membuat data baru maupun menimpa data yang sudah ada, dan penting dalam workflow yang menerima data dari node sebelumnya sebelum dimasukkan ke database. (n8n Docs)

Fungsi praktis:

- memilih field yang dipakai;
- membuang field yang tidak perlu;
- mengganti nama field;
- membuat struktur JSON lebih rapi.

Code

Fungsi:

- validasi sederhana;
- transformasi data;
- logic khusus;
- membuat response custom.

Dokumentasi n8n menjelaskan Code node dapat menjalankan JavaScript atau Python sebagai langkah workflow untuk menambah, menghapus, atau memperbarui item. (n8n Docs)

HTTP Request

Fungsi:

- memanggil API eksternal;
- mengambil data JSON;
- mengirim data ke endpoint lain.

MySQL

Fungsi:

- insert data;
- update data;
- select data;
- menjalankan query SQL.

Dokumentasi n8n menyatakan MySQL node mendukung eksekusi SQL query, insert, dan update row di database MySQL. (n8n Docs)

9.5 Alur Utama: Terima JSON → Validasi → Bersihkan → Simpan ke MySQL

Alur yang akan dibuat:

```
Webhook
  ↓
Edit Fields / Set
  ↓
Code: validasi sederhana
  ↓
MySQL: insert data
  ↓
MySQL: query data
  ↓
Respond to Webhook
```

Data masuk:

```
{
  "nim": "2026001",
  "nama": "Alya Pratama",
  "email": "alya@example.test",
  "prodi": "Informatika",
  "angkatan": 2026,
  "profile": {
    "alamat": {
      "kota": "Tangerang Selatan",
      "provinsi": "Banten"
    },
    "skill": ["Python", "SQL", "Database"]
  }
}
```

Data keluar:

```
{
  "status": "success",
  "message": "Data mahasiswa berhasil disimpan",
  "nim": "2026001"
}
```

Sasaran bab ini: membuat API pendaftaran mahasiswa sederhana tanpa framework berat.

9.6 Menyiapkan Tabel MySQL

Buat tabel untuk workflow.

```
DROP TABLE IF EXISTS n8n_students;

CREATE TABLE n8n_students (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL UNIQUE,
  nama VARCHAR(100) NOT NULL,
  email VARCHAR(150) NOT NULL UNIQUE,
  prodi VARCHAR(100) NOT NULL,
  angkatan INT NOT NULL,
  profile JSON NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Desain ini hybrid:

- data inti menjadi kolom biasa;
- data fleksibel masuk `profile JSON`;
- `nim` dan `email` dibuat unik;
- `created_at` otomatis dari database.

Catatan: field yang sering dicari jangan disembunyikan di JSON. Ini konsisten dengan prinsip Bab 7 tentang *indexing* dan desain hybrid.

9.7 Membuat Webhook Penerima JSON

Langkah di n8n:

- Buat workflow baru.
- Tambahkan node **Webhook**.
- Pilih method `POST`.
- Buat path, misalnya:

/student-register

- Response mode: gunakan response dari node akhir.
- Simpan workflow.
- Aktifkan mode test untuk mendapatkan test URL.

Contoh URL lokal biasanya berbentuk:

```
http://localhost:5678/webhook-test/student-register
```

Catatan: URL test dipakai saat workflow sedang diuji. Untuk workflow aktif, gunakan URL production dari n8n.

9.8 Mengirim Data dari curl

Kirim request:

```
curl -X POST
http://localhost:5678/webhook-test/student-register \
  -H "Content-Type: application/json" \
  -d '{
    "nim": "2026001",
    "nama": "Alya Pratama",
    "email": "alya@example.test",
    "prodi": "Informatika",
    "angkatan": 2026,
    "profile": {
      "alamat": {
        "kota": "Tangerang Selatan",
        "provinsi": "Banten"
      },
      "skill": ["Python", "SQL", "Database"]
    }
  }'
```

Penjelasan:

- `curl`: alat command line untuk mengirim request HTTP.
- `-X POST`: memakai method POST.
- `-H "Content-Type: application/json"`: memberi tahu server bahwa body berisi JSON.
- `-d`: isi data yang dikirim.
- payload harus JSON valid.

Jika Webhook node benar, n8n akan menerima data tersebut.

9.9 Mengirim Data dari Hoppscotch

Hoppscotch adalah ekosistem API development open source yang dapat dipakai untuk membuat dan menguji request API. Situs resminya menyebut Hoppscotch sebagai *open source API development ecosystem*. (hoppscotch.io)

Langkah:

- Buka Hoppscotch.
- Pilih method `POST`.
- Masukkan URL webhook n8n.
- Tambahkan header:

Content-Type: application/json

- Masukkan body JSON.
- Kirim request.
- Lihat response dari n8n.

Contoh body:

```
{
  "nim": "2026002",
  "nama": "Bima Santoso",
  "email": "bima@example.test",
  "prodi": "Sistem Informasi",
  "angkatan": 2026,
  "profile": {
    "alamat": {
      "kota": "Depok",
      "provinsi": "Jawa Barat"
    },
    "skill": ["Python", "API", "Automation"]
  }
}
```

Kegunaan Hoppscotch: mahasiswa bisa melihat request dan response lebih nyaman daripada hanya terminal.

9.10 Membersihkan Field dengan Edit Fields / Set

Setelah Webhook, tambahkan node **Edit Fields / Set**.

Tujuan:

- memilih field yang dibutuhkan;
- memastikan nama field konsisten;
- membuang field liar;
- membentuk payload yang siap diproses.

Field yang disimpan:

```
nim
nama
email
prodi
angkatan
profile
```

Aktifkan opsi **Keep Only Set Fields** agar field yang tidak dipakai dibuang.

Contoh hasil yang diharapkan:

```
{
  "nim": "2026002",
  "nama": "Bima Santoso",
  "email": "bima@example.test",
  "prodi": "Sistem Informasi",
  "angkatan": 2026,
  "profile": {
    "alamat": {
      "kota": "Depok",
      "provinsi": "Jawa Barat"
    },
    "skill": ["Python", "API", "Automation"]
  }
}
```

Prinsip: jangan biarkan payload dari luar masuk apa adanya.

9.11 Validasi Sederhana dengan Code Node

Tambahkan node **Code** setelah Edit Fields.

Logic validasi sederhana:

- `nim` wajib ada;
- `nama` wajib ada;
- `email` wajib ada;
- `prodi` wajib ada;
- `angkatan` harus angka;
- `profile` harus object;
- `profile.alamat.kota` wajib ada;
- `profile.skill` harus array.

Contoh Code node:

```
const item = $input.first().json;
const errors = [];

if (!item.nim) errors.push("nim wajib ada");
if (!item.nama) errors.push("nama wajib ada");
if (!item.email) errors.push("email wajib ada");
if (!item.prodi) errors.push("prodi wajib ada");

if (typeof item.angkatan !== "number") {
  errors.push("angkatan harus angka");
}

if (!item.profile || typeof item.profile !== "object") {
  errors.push("profile harus object");
}

if (!item.profile?.alamat?.kota) {
  errors.push("profile.alamat.kota wajib ada");
}

if (!Array.isArray(item.profile?.skill)) {
  errors.push("profile.skill harus array");
}

if (errors.length > 0) {
  return [
    {
      json: {
        valid: false,
        errors,
```

```
          input: item
        }
      }
    ];
}

return [
  {
    json: {
      valid: true,
      data: item
    }
  }
];
```

Catatan penting: validasi ini hanya sederhana. Untuk validasi yang formal, gunakan JSON Schema seperti Bab 8. n8n bisa menjadi orkestrator; Python bisa menjadi validator yang lebih ketat.

9.12 Memisahkan Data Valid dan Invalid

Tambahkan node **IF**.

Kondisi:

```
valid equals true
```

Cabang:

- **true** → simpan ke MySQL.
- **false** → kirim response error.

Response error:

```
{
  "status": "error",
  "message": "Data tidak valid",
  "errors": [
    "email wajib ada",
    "angkatan harus angka"
  ]
}
```

Prinsip: workflow yang baik tidak hanya sukses saat data benar, tetapi juga memberi pesan jelas saat data salah.

9.13 Insert JSON ke MySQL dari n8n

Tambahkan node **MySQL** pada cabang valid.

Pilih operasi:

- **Insert** jika memakai antarmuka insert row.
- Atau **Execute SQL** jika ingin kontrol query.

Untuk latihan yang transparan, gunakan **Execute SQL**.

Query:

```
INSERT INTO n8n_students
  (nim, nama, email, prodi, angkatan, profile)
VALUES
  (
    '{{ $json.data.nim }}',
    '{{ $json.data.nama }}',
    '{{ $json.data.email }}',
    '{{ $json.data.prodi }}',
    {{ $json.data.angkatan }},
    CAST('{{ JSON.stringify($json.data.profile) }}' AS JSON)
  );
```

Catatan keamanan: query dengan expression seperti ini cocok untuk lab, tetapi untuk sistem serius perlu desain yang lebih aman, validasi ketat, dan pembatasan akses database.

Alternatif yang lebih terstruktur adalah memakai operasi Insert dan memetakan kolom satu per satu:

- `nim` → `{{ $json.data.nim }}`
- `nama` → `{{ $json.data.nama }}`
- `email` → `{{ $json.data.email }}`
- `prodi` → `{{ $json.data.prodi }}`
- `angkatan` → `{{ $json.data.angkatan }}`
- `profile` → `{{ JSON.stringify($json.data.profile) }}`

n8n MySQL node mendukung operasi database seperti Select, Insert, Update, Insert or Update, Delete, dan Execute SQL. (N8N)

9.14 Query MySQL lalu Kirim Response JSON

Setelah insert, tambahkan node MySQL kedua untuk query ulang data.

Query:

```
SELECT
  id,
  nim,
  nama,
  email,
  prodi,
  angkatan,
  profile,
  created_at
FROM n8n_students
WHERE nim = '{{ $json.data.nim }}';
```

Lalu tambahkan node response.

Response sukses:

```
{
  "status": "success",
  "message": "Data mahasiswa berhasil disimpan",
  "data": {
    "nim": "2026001",
    "nama": "Alya Pratama",
    "prodi": "Informatika"
  }
}
```

Jika memakai **Respond to Webhook** node, isi response dapat dibentuk dari output query terakhir.

Prinsip: API yang baik tidak hanya menyimpan data, tetapi memberi response yang jelas.

9.15 Update Data dari n8n

Contoh use case: update `profile` mahasiswa berdasarkan `nim`.

Query:

```
UPDATE n8n_students
SET
  nama = '{{ $json.nama }}',
  prodi = '{{ $json.prodi }}',
  angkatan = {{ $json.angkatan }},
  profile = CAST('{{ JSON.stringify($json.profile) }}' AS
JSON)
WHERE nim = '{{ $json.nim }}';
```

Namun, untuk update field tertentu di JSON, gunakan `JSON_SET`.

Contoh update kota:

```
UPDATE n8n_students
SET profile = JSON_SET(
  profile,
  '$.alamat.kota',
  '{{ $json.kota_baru }}'
)
WHERE nim = '{{ $json.nim }}';
```

Catatan: update sebagian JSON lebih aman daripada menimpa seluruh dokumen, terutama jika hanya satu field berubah.

9.16 Error Handling Sederhana

Error umum dalam workflow:

- JSON tidak valid.
- Field wajib hilang.
- Email duplikat.
- NIM duplikat.
- Koneksi database gagal.
- Query SQL salah.
- Tipe data salah.
- Struktur `profile` tidak sesuai.

Strategi dasar:

- Validasi sebelum MySQL.
- Pisahkan cabang valid dan invalid.
- Gunakan response error yang jelas.
- Simpan error log jika perlu.
- Jangan tampilkan detail sensitif ke client.
- Jangan memberi akses database berlebihan.

Contoh response error:

```
{
  "status": "error",
  "message": "Data tidak dapat diproses",
  "errors": [
    "nim wajib ada",
    "profile.skill harus array"
  ]
}
```

Prinsip: error harus membantu perbaikan, bukan membuat pengguna bingung.

9.17 Export dan Import Workflow JSON

n8n menyimpan workflow dalam format JSON. Dokumentasi n8n menyatakan workflow dapat diekspor sebagai file JSON dan file JSON dapat diimpor kembali ke library n8n. (n8n Docs)

Manfaat export workflow:

- dokumentasi tugas mahasiswa;
- backup workflow;
- berbagi workflow antar tim;
- versioning bersama kode;
- lampiran proyek akhir;
- audit perubahan.

Praktik:

- buka workflow;
- pilih menu workflow;
- pilih download/export;
- simpan sebagai:

```
student-register-workflow.json
```

Import:

- buka n8n;
- pilih import workflow;
- pilih file JSON;
- cek credential;
- jalankan ulang test.

Catatan: credential biasanya tidak dibagikan langsung bersama workflow. Mahasiswa harus mengatur ulang credential MySQL di mesin masing-masing.

9.18 Kapan Logic Dikerjakan di n8n?

Kerjakan logic di n8n jika:

- alurnya sederhana;
- hanya memindahkan data antar sistem;
- hanya mapping field;
- hanya validasi ringan;
- hanya routing berdasarkan kondisi;
- butuh webhook cepat;
- butuh integrasi API dan database;
- butuh prototipe workflow;
- butuh monitoring alur visual.

Contoh cocok untuk n8n:

- menerima data pendaftaran;
- membersihkan nama field;
- insert ke database;
- kirim response;
- kirim notifikasi sederhana;
- export workflow untuk dokumentasi.

9.19 Kapan Logic Dikerjakan di Python?

Kerjakan logic di Python jika:

- validasi kompleks;
- memakai JSON Schema formal;
- transformasi data besar;
- butuh unit test;
- butuh library khusus;
- butuh transaksi kompleks;
- butuh batch processing;
- butuh error handling detail;
- butuh performa lebih terkontrol;
- butuh kode yang mudah diuji otomatis.

Contoh cocok untuk Python:

- validasi JSON Schema seperti Bab 8;
- import ribuan data;
- cleaning data kompleks;
- deduplikasi;
- scoring risiko;
- transformasi nested JSON besar;
- export laporan besar;
- integrasi dengan model analitik.

Strategi terbaik: n8n untuk orkestrasi, Python untuk logic yang berat dan harus diuji.

Hands-on Bab 9 — API Pendaftaran Mahasiswa Sederhana

9.20 Desain Workflow

Workflow akhir:

```
Webhook
  ↓
Edit Fields / Set
  ↓
Code: validasi sederhana
  ↓
IF valid?
  ├── True  → MySQL Insert → MySQL Select → Respond Success
  └── False → Respond Error
```

9.21 Payload Valid

Kirim dari `curl`:

```
curl -X POST
http://localhost:5678/webhook-test/student-register \
  -H "Content-Type: application/json" \
  -d '{
    "nim": "2026003",
    "nama": "Citra Lestari",
    "email": "citra@example.test",
    "prodi": "Teknik Informatika",
    "angkatan": 2026,
    "profile": {
      "alamat": {
        "kota": "Bogor",
        "provinsi": "Jawa Barat"
      },
      "skill": ["SQL", "Python", "Data"]
    }
  }'
```

Response sukses yang diharapkan:

```
{
  "status": "success",
  "message": "Data mahasiswa berhasil disimpan",
  "nim": "2026003"
}
```

9.22 Payload Invalid

Contoh `angkatan` salah tipe:

```
curl -X POST
http://localhost:5678/webhook-test/student-register \
  -H "Content-Type: application/json" \
  -d '{
    "nim": "2026004",
    "nama": "Deni Prakoso",
    "email": "deni@example.test",
    "prodi": "Informatika",
    "angkatan": "dua ribu dua puluh enam",
    "profile": {
      "alamat": {
        "kota": "Serpong",
        "provinsi": "Banten"
      },
      "skill": ["Python"]
    }
  }'
```

Response error yang diharapkan:

```
{
  "status": "error",
  "message": "Data tidak valid",
  "errors": [
    "angkatan harus angka"
  ]
}
```

9.23 Query Cek Data di MySQL

```
SELECT
  id,
  nim,
  nama,
  email,
  prodi,
  angkatan,
  JSON_PRETTY(profile) AS profile,
  created_at
FROM n8n_students
ORDER BY id DESC;
```

Cek kota dari JSON:

```
SELECT
  nim,
  nama,
  profile->>'$.alamat.kota' AS kota
FROM n8n_students;
```

Cek skill pertama:

```
SELECT
  nim,
  nama,
  profile->>'$.skill[0]' AS skill_pertama
FROM n8n_students;
```

9.24 Export Data dari MySQL dengan Python

Buat file `export_n8n_students.py`.

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

try:
    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor(dictionary=True)

    sql = """
        SELECT
          nim,
          nama,
          email,
          prodi,
          angkatan,
          profile,
          created_at
        FROM n8n_students
        ORDER BY nim
    """

    cursor.execute(sql)
    rows = cursor.fetchall()

    result = []

    for row in rows:
        result.append({
            "nim": row["nim"],
            "nama": row["nama"],
            "email": row["email"],
            "prodi": row["prodi"],
            "angkatan": row["angkatan"],
            "profile": json.loads(row["profile"]),
            "created_at": str(row["created_at"])
        })
```

```
    with open("n8n_students_export.json", "w",
encoding="utf-8") as file:
        json.dump(result, file, ensure_ascii=False, indent=2)

    print("Export selesai: n8n_students_export.json")

except Error as err:
    print("Error database:", err)

finally:
    if "cursor" in locals():
        cursor.close()
    if "conn" in locals() and conn.is_connected():
        conn.close()
```

Script ini menunjukkan bahwa workflow n8n tetap dapat digabung dengan Python untuk export, audit, dan laporan.

Mini-Lab Bab 9 — API Pendaftaran Mahasiswa Tanpa Framework Berat

Tujuan

Mahasiswa membuat API sederhana berbasis n8n untuk menerima JSON pendaftaran mahasiswa, membersihkan field, validasi ringan, menyimpan ke MySQL, lalu mengirim response JSON.

Tugas Praktik

Kerjakan:

- Buat tabel `n8n_students`.
- Buat workflow baru di n8n.
- Tambahkan Webhook dengan method `POST`.
- Kirim payload JSON dengan `curl`.
- Uji payload dengan Hoppscotch.
- Tambahkan Edit Fields / Set.
- Buang field yang tidak perlu.
- Tambahkan Code node untuk validasi sederhana.
- Tambahkan IF node untuk memisahkan valid dan invalid.
- Tambahkan MySQL node untuk insert data valid.
- Tambahkan MySQL node untuk query ulang data.
- Tambahkan response sukses.
- Tambahkan response error.
- Export workflow sebagai file JSON.
- Export data dari MySQL dengan Python.

Output yang Diharapkan

File workflow:

```
student-register-workflow.json
```

File export data:

```
n8n_students_export.json
```

Response sukses:

```
{
  "status": "success",
  "message": "Data mahasiswa berhasil disimpan",
  "nim": "2026003"
}
```

Response error:

```
{
  "status": "error",
  "message": "Data tidak valid",
  "errors": [
    "profile.skill harus array"
  ]
}
```

Kesalahan Umum Mahasiswa

Hindari:

- Menganggap n8n menggantikan pemahaman API.
- Tidak memahami struktur item dan JSON di n8n.
- Membiarkan payload masuk tanpa validasi.
- Menyimpan field liar ke database.
- Tidak memisahkan cabang valid dan invalid.
- Tidak memberi response error yang jelas.
- Menulis query tanpa memahami risiko input.
- Menaruh semua logic kompleks di Code node.
- Tidak mengekspor workflow sebagai dokumentasi.
- Tidak mengatur credential database dengan aman.
- Tidak membedakan test webhook dan production webhook.
- Tidak mengecek ulang data di MySQL.

Checklist Akhir Bab

Mahasiswa dinyatakan memahami Bab 9 jika mampu:

- Menjelaskan konsep **workflow automation**.
- Menjelaskan struktur data n8n: item, field, JSON.
- Membuat Webhook penerima JSON.
- Mengirim payload memakai `curl`.
- Menguji request memakai Hoppscotch.
- Membersihkan field dengan Edit Fields / Set.
- Membuat validasi sederhana dengan Code node.
- Memisahkan data valid dan invalid.
- Insert data ke MySQL dari n8n.
- Query data MySQL dari n8n.
- Mengirim response JSON.
- Export workflow sebagai file JSON.
- Menjelaskan kapan logic dikerjakan di n8n.
- Menjelaskan kapan logic dikerjakan di Python.
- Membuat mini-lab API pendaftaran mahasiswa sederhana.

Ringkasan Bab

Bab ini memperkenalkan n8n sebagai alat otomasi open source untuk mempercepat integrasi JSON dan MySQL. Mahasiswa belajar membuat Webhook, memahami struktur item/JSON, membersihkan data dengan Edit Fields, memvalidasi sederhana dengan Code node, menyimpan data ke MySQL, query ulang data, dan mengirim response JSON. n8n mempercepat pembuatan workflow, tetapi tidak menggantikan pemahaman database, JSON, API, dan Python. **Pelajaran utama Bab 9: n8n adalah orkestrator; data tetap harus divalidasi, database tetap harus dirancang, dan logic penting tetap harus bisa diuji.**

Bab 10 — Proyek Akhir: Pipeline JSON, Python, n8n, MySQL, Security

> **Proyek akhir yang baik bukan sekadar berjalan; ia harus tervalidasi, aman, terdokumentasi, bisa diulang, dan bisa dijelaskan.**

Tujuan Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa mampu:

- Mendesain arsitektur integrasi **Python + n8n + MySQL**.
- Membuat pipeline **ETL sederhana**: *extract*, *transform*, *load*.
- Mengambil dan memproses data JSON dari file, webhook, atau API.
- Membuat database hybrid: kolom relasional + kolom JSON.
- Membuat **JSON Schema** sebagai kontrak data.
- Membuat script Python untuk import, validasi, query, dan export.
- Membuat workflow n8n untuk menerima JSON dan menyimpan ke MySQL.
- Membuat query laporan dari data JSON.
- Mengoptimasi satu query dengan *generated column*.
- Membuat user MySQL dengan privilege terbatas.
- Membuat backup dan restore sederhana.
- Menyusun dokumentasi teknis.
- Mempresentasikan hasil proyek secara profesional.

10.1 Posisi Bab 10 dalam Buku

Bab 1 sampai Bab 9 sudah membangun fondasi:

- Bab 1: mengapa JSON penting untuk database modern.
- Bab 2: struktur JSON, validasi dasar, dan *JSON path*.
- Bab 3: CRUD JSON dengan MySQL.
- Bab 4: Python untuk import/export JSON.
- Bab 5: SQLAlchemy sebagai lapisan kode yang lebih rapi.
- Bab 6: query JSON canggih, agregasi, dan `JSON_TABLE`.
- Bab 7: *indexing*, performa, dan skema hybrid.
- Bab 8: JSON Schema sebelum data masuk database.
- Bab 9: n8n untuk automasi JSON dan MySQL.

Alur buku adalah **konsep → dasar JSON → CRUD → Python → SQLAlchemy → query canggih → performance → JSON Schema → n8n → proyek akhir**, sehingga mahasiswa tidak hanya tahu MySQL bisa menyimpan JSON, tetapi mampu membangun sistem nyata yang siap dikembangkan.

Bab 10 menyatukan semuanya menjadi proyek kecil yang bisa dijadikan:

- tugas akhir mata kuliah;
- portofolio mahasiswa;
- demo kerja;
- bahan presentasi;
- fondasi sistem internal sederhana.

10.2 Arsitektur Integrasi Python + n8n + MySQL

Arsitektur proyek akhir:

```
Input JSON
  ↓
n8n Webhook / file JSON / API eksternal
  ↓
Validasi awal
  ↓
Python validator + JSON Schema
  ↓
MySQL hybrid table
  ↓
Query laporan
  ↓
Export JSON / dashboard / presentasi
  ↓
Backup + dokumentasi
```

Peran setiap komponen:

- **n8n**: orkestrasi workflow, menerima webhook, routing data, integrasi cepat.
- **Python**: validasi formal, transformasi data, import/export, laporan.
- **MySQL**: penyimpanan data, query, index, transaksi, backup.
- **JSON Schema**: kontrak struktur data.
- **Generated column**: mempercepat pencarian field penting dari JSON.
- **Dokumentasi**: bukti bahwa proyek bisa dipahami dan diulang.

Dokumentasi n8n menyatakan MySQL node mendukung eksekusi query SQL, insert, dan update row, sehingga n8n cocok menjadi orkestrator data menuju database. (n8n Docs)

10.3 Pilihan Proyek Akhir

Mahasiswa boleh memilih salah satu dari tujuh proyek.

1. Sistem Pendaftaran Mahasiswa Berbasis JSON + MySQL + n8n

Fokus:

- webhook menerima data pendaftaran;
- JSON Schema memvalidasi input;
- MySQL menyimpan data;
- Python membuat laporan;
- n8n memberi response JSON.

Cocok untuk mahasiswa yang ingin memahami API, form, dan database.

2. Inventory Sederhana dengan Spesifikasi Produk Fleksibel

Fokus:

- data inti produk sebagai kolom biasa;
- spesifikasi teknis sebagai JSON;
- query produk berdasarkan kategori, harga, dan tag;
- optimasi tag memakai strategi index.

Cocok untuk mahasiswa yang tertarik e-commerce, UMKM, dan katalog produk.

3. Log Collector JSON untuk Monitoring Aplikasi

Fokus:

- aplikasi mengirim log JSON;
- n8n menerima log;
- Python membersihkan dan memvalidasi;
- MySQL menyimpan log;
- laporan error dibuat otomatis.

Cocok untuk mahasiswa yang tertarik observability dan keamanan aplikasi.

4. Mini ETL Pipeline dari API Publik ke MySQL

Fokus:

- Python mengambil data JSON dari API;
- transformasi struktur;
- validasi field;
- load ke MySQL;
- laporan otomatis.

Cocok untuk mahasiswa data engineering pemula.

5. Dashboard Reporting Berbasis Query JSON

Fokus:

- data JSON disimpan di MySQL;
- query menggunakan `JSON_TABLE`;
- laporan dibuat dari agregasi;
- output JSON siap dashboard.

Cocok untuk mahasiswa yang tertarik analisis data.

6. Sistem Import Data UMKM dari File JSON ke MySQL

Fokus:

- file JSON berisi data pelaku usaha;
- validasi schema;
- deteksi duplikasi;
- simpan ke MySQL;
- export laporan ringkas.

Cocok untuk proyek kampus dan pengabdian masyarakat.

7. Sistem Validasi Data Form Menggunakan JSON Schema + Python

Fokus:

- validasi data ketat;
- error report;
- data valid disimpan;
- data invalid dikembalikan ke penginput.

Cocok untuk mahasiswa yang ingin fokus pada kualitas data.

10.4 Arsitektur Proyek yang Direkomendasikan

Untuk proyek akhir buku ini, gunakan studi kasus:

Sistem Pendaftaran Mahasiswa Berbasis JSON + MySQL + n8n

Alasannya:

- mudah dipahami mahasiswa;
- dekat dengan kehidupan kampus;
- cocok untuk validasi JSON Schema;
- cocok untuk workflow n8n;
- cocok untuk MySQL hybrid;
- cocok untuk query laporan;
- cocok untuk security dasar;
- cocok untuk presentasi.

Arsitektur:

```
Client / curl / Hoppscotch
  ↓ POST JSON
n8n Webhook
  ↓
n8n validasi ringan
  ↓
Python validator JSON Schema
  ↓
MySQL validated_students
  ↓
Python export laporan
  ↓
Backup + dokumentasi
```

10.5 Desain Database Hybrid

Buat database:

```
CREATE DATABASE IF NOT EXISTS final_json_project;
USE final_json_project;
```

Buat tabel utama:

```
DROP TABLE IF EXISTS student_registrations;

CREATE TABLE student_registrations (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL UNIQUE,
  nama VARCHAR(100) NOT NULL,
  email VARCHAR(150) NOT NULL UNIQUE,
  prodi VARCHAR(100) NOT NULL,
  angkatan INT NOT NULL,
  registration_data JSON NOT NULL,
  email_gen VARCHAR(150)
    GENERATED ALWAYS AS (registration_data->>'$.email')
STORED,
  kota_gen VARCHAR(100)
    GENERATED ALWAYS AS (registration_data->>'$.alamat.kota')
STORED,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  INDEX idx_email_gen (email_gen),
  INDEX idx_kota_gen (kota_gen)
);
```

Penjelasan:

- `nim`, `nama`, `email`, `prodi`, `angkatan`: data inti.
- `registration_data`: data fleksibel lengkap.
- `email_gen`: field JSON yang diekstrak untuk pencarian.
- `kota_gen`: field JSON yang diekstrak untuk laporan.
- `idx_email_gen`, `idx_kota_gen`: index untuk query cepat.

Desain ini mengikuti prinsip Bab 7: **data penting menjadi kolom biasa, data fleksibel masuk JSON, dan data yang sering dicari wajib punya strategi index**.

10.6 Tabel Logging dan Audit Trail Sederhana

Proyek akhir harus punya jejak proses. Buat tabel log.

```
DROP TABLE IF EXISTS workflow_events;

CREATE TABLE workflow_events (
  id INT AUTO_INCREMENT PRIMARY KEY,
  event_type VARCHAR(50) NOT NULL,
  source_name VARCHAR(100) NOT NULL,
  status VARCHAR(30) NOT NULL,
  event_data JSON,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Contoh event:

- `webhook_received`
- `validation_success`
- `validation_failed`
- `db_insert_success`
- `db_insert_failed`
- `export_report_success`

Audit trail sederhana membantu menjawab:

- data datang dari mana;
- kapan diproses;
- valid atau tidak;
- error apa yang terjadi;
- siapa atau workflow apa yang menjalankan.

Disini juga menempatkan logging, masking data sensitif, audit trail, backup, dan validasi sebagai bagian checklist keamanan proyek.

10.7 JSON Schema Proyek Akhir

Buat file `schema/student_registration.schema.json`.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Student Registration",
  "type": "object",
  "required": ["nim", "nama", "email", "prodi", "angkatan",
"alamat", "skill"],
  "properties": {
    "nim": {
      "type": "string",
      "minLength": 5,
      "maxLength": 20
    },
    "nama": {
      "type": "string",
      "minLength": 3,
      "maxLength": 100
    },
    "email": {
      "type": "string",
      "format": "email"
    },
    "prodi": {
      "type": "string",
      "enum": ["Informatika", "Sistem Informasi", "Teknik
Informatika"]
    },
    "angkatan": {
      "type": "integer",
      "minimum": 2020,
      "maximum": 2030
    },
    "alamat": {
      "type": "object",
      "required": ["kota", "provinsi"],
      "properties": {
        "kota": {
          "type": "string",
          "minLength": 3
        },
        "provinsi": {
          "type": "string",
          "minLength": 3
        }
      },
```

```
      "additionalProperties": false
    },
    "skill": {
      "type": "array",
      "minItems": 1,
      "items": {
        "type": "string",
        "minLength": 2
      }
    }
  },
  "additionalProperties": false
}
```

JSON Schema memberi struktur, constraint, dan tipe data untuk dokumen JSON; dokumentasi JSON Schema menjelaskan bahwa schema mendeskripsikan struktur, constraint, dan tipe data dari JSON. (JSON Schema)

Itelman & Viotti juga menekankan pentingnya schema untuk mengurangi ambiguitas data dan menyelaraskan konsep, struktur, serta validasi data.

10.8 Struktur Folder Proyek

Gunakan struktur sederhana:

```
final-json-project/
├── schema/
│   └── student_registration.schema.json
├── data/
│   ├── students_valid.json
│   └── students_invalid.json
├── scripts/
│   ├── validate_students.py
│   ├── import_students.py
│   ├── export_report.py
│   └── backup_database.sh
├── sql/
│   ├── 01_create_tables.sql
│   ├── 02_create_user.sql
│   └── 03_report_queries.sql
├── workflows/
│   └── student_registration_n8n.json
├── reports/
│   ├── valid_students_report.json
│   └── invalid_students_report.json
└── README.md
```

Prinsip: proyek yang baik bukan hanya script. Proyek harus punya struktur, data contoh, SQL, workflow, laporan, dan dokumentasi.

10.9 Script Python Validasi

Install pustaka:

```
python3 -m venv venv
source venv/bin/activate
pip install jsonschema mysql-connector-python
```

Buat file `scripts/validate_students.py`.

```
import json
from jsonschema import Draft202012Validator, FormatChecker

def load_json(path):
    with open(path, "r", encoding="utf-8") as file:
        return json.load(file)

def validate_records(records, schema):
    validator = Draft202012Validator(
        schema,
        format_checker=FormatChecker()
    )

    valid = []
    invalid = []

    for index, record in enumerate(records):
        errors = sorted(
            validator.iter_errors(record),
            key=lambda error: list(error.path)
        )

        if errors:
            invalid.append({
                "row": index + 1,
                "nim": record.get("nim"),
                "errors": [
                    {
                        "path": ".".join(str(item) for item
in error.path),
                        "message": error.message
                    }
                    for error in errors
                ]
            })
        else:
            valid.append(record)
```

```
    return valid, invalid


if __name__ == "__main__":
    schema =
load_json("schema/student_registration.schema.json")
    records = load_json("data/students_valid.json")

    valid, invalid = validate_records(records, schema)

    with open("reports/valid_students_report.json", "w",
encoding="utf-8") as file:
        json.dump(valid, file, ensure_ascii=False, indent=2)

    with open("reports/invalid_students_report.json", "w",
encoding="utf-8") as file:
        json.dump(invalid, file, ensure_ascii=False,
indent=2)

    print("Valid:", len(valid))
    print("Invalid:", len(invalid))
```

Penjelasan:

- `Draft202012Validator`: memakai JSON Schema Draft 2020-12.
- `FormatChecker`: mengecek format seperti email.
- `iter_errors`: mengumpulkan semua error.
- data valid dan invalid dipisahkan.
- laporan invalid disimpan, tidak dibuang diam-diam.

10.10 Script Python Import ke MySQL

Buat file `scripts/import_students.py`.

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "student_app",
    "password": "GantiPasswordAplikasi",
    "database": "final_json_project"
}


def load_json(path):
    with open(path, "r", encoding="utf-8") as file:
        return json.load(file)


def log_event(cursor, event_type, source_name, status, event_data):
    sql = """
        INSERT INTO workflow_events
          (event_type, source_name, status, event_data)
        VALUES
          (%s, %s, %s, %s)
    """

    cursor.execute(sql, (
        event_type,
        source_name,
        status,
        json.dumps(event_data, ensure_ascii=False)
    ))


def import_students(records):
    conn = None
    cursor = None

    try:
        conn = mysql.connector.connect(**DB_CONFIG)
        cursor = conn.cursor(prepared=True)
        conn.start_transaction()

        sql = """
            INSERT INTO student_registrations
              (nim, nama, email, prodi, angkatan, registration_data)
            VALUES
              (%s, %s, %s, %s, %s, %s)
        """
```

```
            for record in records:
                cursor.execute(sql, (
                    record["nim"],
                    record["nama"],
                    record["email"],
                    record["prodi"],
                    record["angkatan"],
                    json.dumps(record, ensure_ascii=False)
                ))

                log_event(
                    cursor,
                    "db_insert_success",
                    "python_import",
                    "success",
                    {"nim": record["nim"]}
                )

            conn.commit()
            print("Import berhasil:", len(records), "data")

        except Error as err:
            if conn:
                conn.rollback()
            print("Error database:", err)

        finally:
            if cursor:
                cursor.close()
            if conn and conn.is_connected():
                conn.close()


    if __name__ == "__main__":
        valid_records = load_json("reports/valid_students_report.json")
        import_students(valid_records)
```

Script ini memakai parameter `%s` dan prepared cursor. MySQL menyatakan prepared statement membantu melindungi dari *SQL injection* karena nilai parameter dipisahkan dari statement SQL. (MySQL Developer Zone)

10.11 Workflow n8n Proyek Akhir

Workflow n8n:

```
Webhook
  ↓
Edit Fields
  ↓
Code: validasi ringan
  ↓
MySQL: Insert workflow_events
  ↓
MySQL: Insert student_registrations
  ↓
MySQL: Select by nim
  ↓
Respond to Webhook
```

Payload masuk:

```
{
  "nim": "2026001",
  "nama": "Alya Pratama",
  "email": "alya@example.test",
  "prodi": "Informatika",
  "angkatan": 2026,
  "alamat": {
    "kota": "Tangerang Selatan",
    "provinsi": "Banten"
  },
  "skill": ["Python", "SQL", "JSON"]
}
```

Response sukses:

```
{
  "status": "success",
  "message": "Data pendaftaran berhasil diterima",
  "nim": "2026001"
}
```

Response gagal:

```
{
  "status": "error",
  "message": "Data tidak valid",
  "errors": [
    "email wajib ada",
    "angkatan harus angka"
  ]
}
```

Workflow harus diekspor sebagai file JSON. n8n mendukung export dan import workflow sebagai file JSON, sehingga workflow dapat dijadikan bagian dokumentasi proyek. (n8n Docs)

10.12 Query Laporan dari MySQL

Laporan jumlah mahasiswa per prodi

```
SELECT
  prodi,
  COUNT(*) AS jumlah_mahasiswa
FROM student_registrations
GROUP BY prodi
ORDER BY jumlah_mahasiswa DESC;
```

Laporan mahasiswa per kota dari JSON

```
SELECT
  kota_gen AS kota,
  COUNT(*) AS jumlah_mahasiswa
FROM student_registrations
GROUP BY kota_gen
ORDER BY jumlah_mahasiswa DESC;
```

Laporan skill dalam JSON dengan JSON_TABLE

```
SELECT
  jt.skill,
  COUNT(*) AS jumlah
FROM student_registrations s
JOIN JSON_TABLE(
  s.registration_data,
  '$.skill[*]'
  COLUMNS (
    skill VARCHAR(100) PATH '$'
  )
) AS jt
GROUP BY jt.skill
ORDER BY jumlah DESC;
```

Stokes menjelaskan bahwa `JSON_TABLE()` mengubah data JSON menjadi bentuk tabel relasional sementara, sehingga dapat diproses dengan kekuatan SQL seperti `WHERE`, `GROUP BY`, dan agregasi.

10.13 Export Laporan dengan Python

Buat file `scripts/export_report.py`.

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "student_app",
    "password": "GantiPasswordAplikasi",
    "database": "final_json_project"
}


def run_query(cursor, sql):
    cursor.execute(sql)
    return cursor.fetchall()


try:
    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor(dictionary=True)

    report_by_prodi = run_query(cursor, """
        SELECT prodi, COUNT(*) AS jumlah_mahasiswa
        FROM student_registrations
        GROUP BY prodi
        ORDER BY jumlah_mahasiswa DESC
    """)

    report_by_city = run_query(cursor, """
        SELECT kota_gen AS kota, COUNT(*) AS jumlah_mahasiswa
        FROM student_registrations
        GROUP BY kota_gen
        ORDER BY jumlah_mahasiswa DESC
    """)

    report_by_skill = run_query(cursor, """
        SELECT jt.skill, COUNT(*) AS jumlah
        FROM student_registrations s
        JOIN JSON_TABLE(
          s.registration_data,
          '$.skill[*]'
          COLUMNS (
            skill VARCHAR(100) PATH '$'
          )
        ) AS jt
        GROUP BY jt.skill
        ORDER BY jumlah DESC
    """)
```

```
        result = {
            "status": "success",
            "reports": {
                "by_prodi": report_by_prodi,
                "by_city": report_by_city,
                "by_skill": report_by_skill
            }
        }

        with open("reports/final_report.json", "w", encoding="utf-8") as
    file:
            json.dump(result, file, ensure_ascii=False, indent=2,
    default=str)

        print("Export laporan selesai")

    except Error as err:
        print("Error database:", err)

    finally:
        if "cursor" in locals():
            cursor.close()
        if "conn" in locals() and conn.is_connected():
            conn.close()
```

Output:

```
{
  "status": "success",
  "reports": {
    "by_prodi": [
      {
        "prodi": "Informatika",
        "jumlah_mahasiswa": 10
      }
    ],
    "by_city": [
      {
        "kota": "Tangerang Selatan",
        "jumlah_mahasiswa": 5
      }
    ],
    "by_skill": [
      {
        "skill": "Python",
        "jumlah": 12
      }
    ]
  }
}
```

10.14 Optimasi Satu Query dengan Generated Column

Query sebelum optimasi:

```
EXPLAIN
SELECT
  nim,
  nama,
  registration_data->>'$.alamat.kota' AS kota
FROM student_registrations
WHERE registration_data->>'$.alamat.kota' = 'Tangerang
Selatan';
```

Query sesudah optimasi:

```
EXPLAIN
SELECT
  nim,
  nama,
  kota_gen AS kota
FROM student_registrations
WHERE kota_gen = 'Tangerang Selatan';
```

Mahasiswa wajib membandingkan:

- `possible_keys`;
- `key`;
- `rows`;
- `type`;
- hasil query.

Stokes menunjukkan bahwa tanpa index, query JSON dapat menjadi *full table scan* dengan `type: ALL` dan `key: NULL`, sehingga `EXPLAIN` penting untuk melihat rencana eksekusi.

10.15 User MySQL dengan Privilege Terbatas

Jangan memakai user administratif untuk aplikasi.

Buat user aplikasi:

```
CREATE USER 'student_app'@'localhost'
IDENTIFIED BY 'GantiPasswordAplikasi';

GRANT SELECT, INSERT, UPDATE
ON final_json_project.student_registrations
TO 'student_app'@'localhost';

GRANT SELECT, INSERT
ON final_json_project.workflow_events
TO 'student_app'@'localhost';

FLUSH PRIVILEGES;
```

Prinsip:

- user aplikasi tidak perlu `DROP`;
- user aplikasi tidak perlu `CREATE`;
- user aplikasi tidak perlu akses semua database;
- privilege mengikuti kebutuhan minimum.

Keamanan dasar: semakin kecil hak akses, semakin kecil kerusakan jika credential bocor.

10.16 Prepared Statement dan Pencegahan SQL Injection

Prepared statement wajib dipakai untuk input dari luar.

Contoh buruk:

```
sql = "SELECT * FROM student_registrations WHERE nim = '" +
nim + "'"
cursor.execute(sql)
```

Contoh benar:

```
sql = "SELECT * FROM student_registrations WHERE nim = %s"
cursor.execute(sql, (nim,))
```

Panduan keamanan client MySQL menyarankan penggunaan prepared statement dan menerima data hanya melalui placeholder untuk statement SQL. (MySQL Developer Zone)

Aturan proyek: semua script Python yang menerima input harus memakai parameter query, bukan penggabungan string.

10.17 Backup dan Restore Sederhana

Backup database:

```
mysqldump -u root -p final_json_project >
backup_final_json_project.sql
```

Restore database:

```
mysql -u root -p final_json_project <
backup_final_json_project.sql
```

Dokumentasi MySQL menjelaskan bahwa `mysqldump` membuat logical backup berupa SQL statement yang dapat dijalankan ulang untuk mereproduksi definisi object dan data tabel. (MySQL Developer Zone)

Dokumentasi MySQL juga menegaskan bahwa backup berguna untuk pemulihan data, membuat salinan untuk eksperimen, dan menguji kompatibilitas perubahan. (MySQL Developer Zone)

Aturan proyek: presentasi akhir belum lengkap jika belum ada file backup.

10.18 Versioning Struktur JSON

Struktur JSON bisa berubah. Karena itu, proyek harus punya versi schema.

Contoh tambahkan versi:

```
{
  "schema_version": "1.0.0",
  "nim": "2026001",
  "nama": "Alya Pratama",
  "email": "alya@example.test"
}
```

Aturan sederhana:

- `1.0.0`: versi awal stabil.
- `1.1.0`: penambahan field opsional.
- `2.0.0`: perubahan besar yang tidak kompatibel.
- schema lama jangan dihapus jika data lama masih ada.
- dokumentasikan perubahan schema.

Itelman & Viotti menyarankan schema diperlakukan sebagai resource yang stabil, dan perubahan signifikan sebaiknya dipublikasikan sebagai versi baru.

Pesan strategis: data yang tidak punya versi akan sulit dipelihara saat sistem berkembang.

10.19 Etika Penggunaan Data

Proyek akhir sering memakai data mahasiswa, pelanggan, UMKM, atau log aplikasi. Data seperti ini harus diperlakukan hati-hati.

Prinsip etika:

- gunakan data dummy untuk demo publik;
- jangan tampilkan data pribadi asli;
- masking email dan nomor kontak jika perlu;
- jangan upload credential ke repository;
- jangan bagikan dump database berisi data sensitif;
- batasi akses database;
- dokumentasikan asal data;
- jelaskan tujuan pemakaian data;
- hapus data yang tidak diperlukan.

Contoh masking email:

```
alya@example.test → a***@example.test
```

Contoh masking nomor:

```
081200000001 → 0812******01
```

Prinsip etika: kemampuan teknis harus diikuti tanggung jawab data.

10.20 AI-assisted Query: Tren Masa Depan, Tetap Harus Diverifikasi

AI dapat membantu:

- menyusun query awal;
- menjelaskan error;
- membuat draft dokumentasi;
- membuat contoh JSON;
- membantu membuat schema;
- menyarankan laporan.

Tetapi AI tidak boleh dipercaya mentah-mentah.

Aturan:

- cek query dengan `EXPLAIN`;
- uji di database dummy;
- validasi hasil dengan data contoh;
- jangan kirim data sensitif;
- pisahkan fakta dan dugaan;
- jangan pakai query yang tidak dipahami;
- simpan bukti hasil uji.

Disini juga mengingatkan bahwa *AI-assisted query* adalah tren masa depan, tetapi tetap harus diverifikasi.

Pesan kritis: AI boleh membantu berpikir, tetapi tanggung jawab tetap pada manusia.

Hands-on Akhir Bab 10

10.21 Langkah 1 — Buat Database dan Tabel

Jalankan:

```
mysql -u root -p < sql/01_create_tables.sql
```

Isi `sql/01_create_tables.sql`:

```
CREATE DATABASE IF NOT EXISTS final_json_project;
USE final_json_project;

DROP TABLE IF EXISTS workflow_events;
DROP TABLE IF EXISTS student_registrations;

CREATE TABLE student_registrations (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL UNIQUE,
  nama VARCHAR(100) NOT NULL,
  email VARCHAR(150) NOT NULL UNIQUE,
  prodi VARCHAR(100) NOT NULL,
  angkatan INT NOT NULL,
  registration_data JSON NOT NULL,
  email_gen VARCHAR(150)
    GENERATED ALWAYS AS (registration_data->>'$.email') STORED,
  kota_gen VARCHAR(100)
    GENERATED ALWAYS AS (registration_data->>'$.alamat.kota') STORED,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  INDEX idx_email_gen (email_gen),
  INDEX idx_kota_gen (kota_gen)
);

CREATE TABLE workflow_events (
  id INT AUTO_INCREMENT PRIMARY KEY,
  event_type VARCHAR(50) NOT NULL,
  source_name VARCHAR(100) NOT NULL,
  status VARCHAR(30) NOT NULL,
  event_data JSON,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

10.22 Langkah 2 — Buat User Aplikasi

Isi `sql/02_create_user.sql`:

```
CREATE USER IF NOT EXISTS 'student_app'@'localhost'
IDENTIFIED BY 'GantiPasswordAplikasi';

GRANT SELECT, INSERT, UPDATE
ON final_json_project.student_registrations
TO 'student_app'@'localhost';

GRANT SELECT, INSERT
ON final_json_project.workflow_events
TO 'student_app'@'localhost';

FLUSH PRIVILEGES;
```

Jalankan:

```
mysql -u root -p < sql/02_create_user.sql
```

10.23 Langkah 3 — Buat Data Contoh

File `data/students_valid.json`:

```
[
  {
    "nim": "2026001",
    "nama": "Alya Pratama",
    "email": "alya@example.test",
    "prodi": "Informatika",
    "angkatan": 2026,
    "alamat": {
      "kota": "Tangerang Selatan",
      "provinsi": "Banten"
    },
    "skill": ["Python", "SQL", "JSON"]
  },
  {
    "nim": "2026002",
    "nama": "Bima Santoso",
    "email": "bima@example.test",
    "prodi": "Sistem Informasi",
    "angkatan": 2026,
    "alamat": {
      "kota": "Depok",
      "provinsi": "Jawa Barat"
    },
    "skill": ["n8n", "API", "MySQL"]
  }
]
```

File `data/students_invalid.json`:

```
[
  {
    "nim": "2026003",
    "nama": "Citra",
    "email": "bukan-email",
    "prodi": "Informatika",
    "angkatan": "dua ribu dua puluh enam",
    "alamat": {
      "kota": "Bogor",
      "provinsi": "Jawa Barat"
    },
    "skill": ["Python"]
  }
]
```

10.24 Langkah 4 — Jalankan Validasi dan Import

Validasi:

```
python scripts/validate_students.py
```

Import:

```
python scripts/import_students.py
```

Cek database:

```
SELECT
  nim,
  nama,
  email,
  prodi,
  angkatan,
  kota_gen,
  JSON_PRETTY(registration_data) AS data_json
FROM student_registrations;
```

10.25 Langkah 5 — Buat Workflow n8n

Workflow minimal:

- **Webhook**: `POST /student-register`
- **Edit Fields**: ambil `nim`, `nama`, `email`, `prodi`, `angkatan`, `alamat`, `skill`
- **Code**: validasi ringan
- **IF**: valid atau invalid
- **MySQL Insert**: simpan data valid
- **MySQL Insert**: simpan event log
- **Respond to Webhook**: kirim response

Kirim uji:

```
curl -X POST
http://localhost:5678/webhook-test/student-register \
  -H "Content-Type: application/json" \
  -d '{
    "nim": "2026004",
    "nama": "Deni Prakoso",
    "email": "deni@example.test",
    "prodi": "Teknik Informatika",
    "angkatan": 2026,
    "alamat": {
      "kota": "Serpong",
      "provinsi": "Banten"
    },
    "skill": ["Python", "Database"]
  }'
```

10.26 Langkah 6 — Buat Query Laporan

Isi `sql/03_report_queries.sql`:

```
USE final_json_project;

SELECT
  prodi,
  COUNT(*) AS jumlah_mahasiswa
FROM student_registrations
GROUP BY prodi
ORDER BY jumlah_mahasiswa DESC;

SELECT
  kota_gen AS kota,
  COUNT(*) AS jumlah_mahasiswa
FROM student_registrations
GROUP BY kota_gen
ORDER BY jumlah_mahasiswa DESC;

SELECT
  jt.skill,
  COUNT(*) AS jumlah
FROM student_registrations s
JOIN JSON_TABLE(
  s.registration_data,
  '$.skill[*]'
  COLUMNS (
    skill VARCHAR(100) PATH '$'
  )
) AS jt
GROUP BY jt.skill
ORDER BY jumlah DESC;
```

Jalankan:

```
mysql -u student_app -p final_json_project <
sql/03_report_queries.sql
```

10.27 Langkah 7 — Backup Database

Buat file `scripts/backup_database.sh`:

```
#!/bin/bash

BACKUP_FILE="backup_final_json_project_$(date
+%Y%m%d_%H%M%S).sql"

mysqldump -u root -p final_json_project > "$BACKUP_FILE"

echo "Backup selesai: $BACKUP_FILE"
```

Jalankan:

```
chmod +x scripts/backup_database.sh
./scripts/backup_database.sh
```

10.28 Langkah 8 — Dokumentasi Teknis

File `README.md` minimal berisi:

```
# Proyek Akhir JSON, Python, n8n, MySQL

## 1. Deskripsi
Sistem ini menerima data pendaftaran mahasiswa dalam format
JSON, memvalidasi data, menyimpan ke MySQL, membuat laporan,
dan menyediakan workflow n8n.

## 2. Arsitektur
Client → n8n Webhook → Validasi → MySQL → Python Export →
Laporan

## 3. Struktur Folder
Jelaskan folder schema, data, scripts, sql, workflows,
reports.

## 4. Cara Instalasi
Jelaskan instalasi Python package, pembuatan database, dan
import workflow n8n.

## 5. Cara Menjalankan
Jelaskan validasi, import, test webhook, export laporan, dan
backup.

## 6. Desain Database
Jelaskan tabel student_registrations dan workflow_events.

## 7. JSON Schema
Jelaskan field wajib, tipe data, enum, dan validasi email.

## 8. Keamanan
Prepared statement, privilege terbatas, masking data, backup.

## 9. Hasil Laporan
Jumlah mahasiswa per prodi, kota, dan skill.

## 10. Lesson Learned
Apa yang berhasil, kendala, dan pengembangan berikutnya.
```

10.29 Checklist Keamanan Proyek

Pastikan:

- **Prepared statement** dipakai di Python.
- **User MySQL aplikasi** tidak memakai hak administratif.
- **Password tidak ditulis di README**.
- **Data dummy** dipakai untuk demo.
- **JSON Schema** dipakai sebelum insert.
- **Data invalid** punya laporan error.
- **Workflow n8n** diekspor.
- **Backup database** dibuat.
- **Generated column** dipakai untuk field yang sering dicari.
- **EXPLAIN** dipakai minimal pada satu query.
- **Audit trail** disimpan ke `workflow_events`.
- **Dokumentasi teknis** lengkap.

10.30 Rubrik Penilaian Proyek Akhir

Aspek	Bobot
Desain database hybrid	15%
Kualitas struktur JSON	10%
JSON Schema dan validasi	15%
Script Python import/export	15%
Workflow n8n	15%
Query laporan	10%
Keamanan dasar	10%
Backup dan dokumentasi	5%
Presentasi	5%

Rubrik ini sesuai arah lampiran attached yang menilai desain database, kualitas struktur JSON, validasi JSON Schema, keamanan, workflow n8n, script Python, optimasi query, dokumentasi, dan presentasi.

10.31 Format Presentasi Hasil Proyek

Gunakan struktur 7 slide:

1. **Masalah**
    - Data apa yang diproses?
    - Kenapa JSON dipakai?
2. **Arsitektur**
    - Python + n8n + MySQL.
    - Alur data end-to-end.
3. **Desain Database**
    - Tabel relasional.
    - Kolom JSON.
    - Generated column.
4. **Validasi**
    - JSON Schema.
    - Data valid dan invalid.
5. **Workflow**
    - Webhook.
    - Validasi.
    - Insert.
    - Response.
6. **Laporan**
    - Query per prodi.
    - Query per kota.
    - Query skill dengan `JSON_TABLE`.
7. **Security dan Lesson Learned**
    - Prepared statement.
    - Privilege terbatas.
    - Backup.
    - Etika data.
    - Rencana pengembangan.

Mini-Lab Bab 10 — Proyek Akhir End-to-End

Tujuan

Mahasiswa membangun pipeline nyata untuk pendaftaran mahasiswa berbasis JSON.

Output Wajib

Mahasiswa wajib mengumpulkan:

- file SQL pembuatan tabel;
- JSON Schema;
- file JSON contoh valid dan invalid;
- script Python validasi;
- script Python import;
- script Python export laporan;
- workflow n8n JSON;
- hasil query laporan;
- file backup `.sql`;
- dokumentasi `README.md`;
- presentasi ringkas.

Kriteria Lulus Minimal

Proyek dianggap selesai jika:

- data JSON valid bisa masuk MySQL;
- data JSON invalid ditolak;
- workflow n8n bisa menerima request;
- Python bisa export laporan;
- ada minimal satu query dengan `JSON_TABLE`;
- ada minimal satu *generated column*;
- ada user MySQL dengan privilege terbatas;
- ada backup database;
- ada dokumentasi teknis.

Kesalahan Umum Mahasiswa

Hindari:

- membuat proyek hanya berisi potongan latihan;
- tidak punya arsitektur jelas;
- semua data dimasukkan ke satu kolom JSON;
- tidak memakai JSON Schema;
- tidak memakai prepared statement;
- memakai user database dengan hak terlalu besar;
- tidak membuat laporan error;
- tidak membuat backup;
- tidak memakai data dummy untuk presentasi;
- tidak mengekspor workflow n8n;
- tidak membuat README;
- memakai AI untuk membuat query tetapi tidak mengujinya;
- tidak bisa menjelaskan isi kode sendiri.

Checklist Akhir Bab

Mahasiswa dinyatakan memahami Bab 10 jika mampu:

- Menjelaskan arsitektur proyek akhir.
- Mendesain database hybrid.
- Membuat tabel dengan kolom JSON.
- Membuat JSON Schema.
- Membuat script Python validasi.
- Membuat script Python import.
- Membuat script Python export laporan.
- Membuat workflow n8n.
- Mengirim payload JSON via webhook.
- Membuat query laporan dari MySQL.
- Menggunakan `JSON_TABLE`.
- Menggunakan *generated column*.
- Menggunakan prepared statement.
- Membuat user MySQL dengan privilege terbatas.
- Membuat backup dan restore sederhana.
- Membuat dokumentasi teknis.
- Mempresentasikan hasil proyek secara jelas.

Ringkasan Bab

Bab 10 mengikat seluruh isi buku menjadi proyek akhir yang nyata. Mahasiswa membangun pipeline JSON dari input ke validasi, dari Python ke MySQL, dari n8n ke workflow, dari query ke laporan, dan dari sistem berjalan ke dokumentasi teknis. Proyek ini menegaskan bahwa kemampuan teknis tidak cukup jika data tidak divalidasi, akses tidak dibatasi, query tidak diuji, dan hasil tidak terdokumentasi. **Pelajaran utama Bab 10: sistem data yang baik harus berjalan, aman, tervalidasi, bisa diaudit, bisa dibackup, dan bisa dijelaskan dengan jernih.**

Lampiran A — Cheat Sheet JSON MySQL

Lampiran ini berisi fungsi JSON MySQL yang paling sering dipakai dalam buku. Fungsi-fungsi ini dipakai untuk **validasi, membaca, mengubah, menghapus, mencari, mengagregasi, dan mengubah JSON menjadi tabel**. MySQL menyediakan referensi resmi fungsi JSON, termasuk fungsi ekstraksi, pencarian, modifikasi, dan agregasi JSON seperti `JSON_ARRAYAGG()` dan `JSON_OBJECTAGG()`. (MySQL Developer Zone)

A.1 Fungsi Validasi dan Tipe Data

Fungsi	Kegunaan	Contoh
`JSON_VALID(json_doc)`	Mengecek apakah JSON valid	`SELECT JSON_VALID('{"nama":"Alya"}');`
`JSON_TYPE(json_val)`	Melihat tipe nilai JSON	`SELECT JSON_TYPE('["Python","SQL"]');`

Contoh:

```
SELECT JSON_VALID('{"nim":"2026001","nama":"Alya"}') AS valid_json;
SELECT JSON_TYPE('{"nim":"2026001"}') AS tipe_json;
```

Makna hasil:

- `JSON_VALID = 1` berarti JSON valid.
- `JSON_VALID = 0` berarti JSON tidak valid.
- `JSON_TYPE` dapat menghasilkan `OBJECT`, `ARRAY`, `STRING`, `INTEGER`, dan tipe lain.

A.2 Fungsi Membaca JSON

Fungsi / Operator	Kegunaan	Contoh
`JSON_EXTRACT(json_doc, path)`	Mengambil nilai dari path tertentu	`JSON_EXTRACT(profile, '$.alamat.kota')`
`->`	Alias ringkas `JSON_EXTRACT`	`profile->'$.alamat.kota'`
`->>`	Ambil nilai tanpa quote JSON	`profile->>'$.alamat.kota'`

Contoh:

```
SELECT
  nim,
  nama,
  profile->>'$.alamat.kota' AS kota
FROM mahasiswa_profile;
```

Catatan:

- Gunakan `->` jika ingin hasil tetap berbentuk JSON.
- Gunakan `->>` jika ingin hasil teks bersih.

A.3 Fungsi Mengubah JSON

Fungsi	Kegunaan	Contoh
`JSON_SET`	Mengubah nilai lama atau menambah path baru	`JSON_SET(profile, '$.alamat.kota', 'Depok')`
`JSON_REMOVE`	Menghapus field atau elemen JSON	`JSON_REMOVE(profile, '$.alamat.kodepos')`

Contoh update:

```
UPDATE mahasiswa_profile
SET profile = JSON_SET(profile, '$.alamat.kota', 'Depok')
WHERE nim = '2026001';
```

Contoh hapus field:

```
UPDATE mahasiswa_profile
SET profile = JSON_REMOVE(profile, '$.alamat.kodepos')
WHERE nim = '2026001';
```

A.4 Fungsi Pencarian JSON

Fungsi / Operator	Kegunaan	Contoh
`JSON_CONTAINS`	Mengecek apakah JSON mengandung nilai tertentu	`JSON_CONTAINS(profile->'$.skill', JSON_QUOTE('Python'))`
`JSON_OVERLAPS`	Mengecek apakah dua JSON memiliki irisan nilai	`JSON_OVERLAPS(profile->'$.skill', JSON_ARRAY('Python','SQL'))`
`MEMBER OF`	Mengecek apakah nilai adalah anggota array JSON	`'Python' MEMBER OF (profile->'$.skill')`

Contoh:

```
SELECT
  nim,
  nama
FROM mahasiswa_profile
WHERE JSON_CONTAINS(profile->'$.skill',
JSON_QUOTE('Python'));
```

Contoh `MEMBER OF`:

```
SELECT
  nim,
  nama
FROM mahasiswa_profile
WHERE 'Python' MEMBER OF (profile->'$.skill');
```

A.5 Fungsi Agregasi JSON

Fungsi	Kegunaan	Contoh
`JSON_ARRAYAGG`	Mengubah hasil query menjadi array JSON	`JSON_ARRAYAGG(nama)`
`JSON_OBJECTAGG`	Mengubah hasil query menjadi object JSON key-value	`JSON_OBJECTAGG(prodi, jumlah)`

Contoh array:

```
SELECT
  prodi,
  JSON_ARRAYAGG(nama) AS daftar_mahasiswa
FROM mahasiswa_profile
GROUP BY prodi;
```

Contoh object:

```
SELECT
  JSON_OBJECTAGG(prodi, jumlah) AS statistik_prodi
FROM (
  SELECT prodi, COUNT(*) AS jumlah
  FROM validated_students
  GROUP BY prodi
) AS summary;
```

A.6 Mengubah JSON Menjadi Tabel dengan JSON_TABLE

`JSON_TABLE` digunakan untuk mengubah array JSON menjadi baris dan kolom. MySQL mendokumentasikan `JSON_TABLE()` sebagai fungsi untuk mengekstrak data dari dokumen JSON dan mengembalikannya sebagai tabel relasional dengan kolom yang ditentukan. (MySQL Developer Zone)

Contoh:

```
SELECT
  o.order_code,
  jt.sku,
  jt.nama_produk,
  jt.qty,
  jt.harga
FROM orders_json o
JOIN JSON_TABLE(
  o.order_data,
  '$.items[*]'
  COLUMNS (
    sku VARCHAR(50) PATH '$.sku',
    nama_produk VARCHAR(100) PATH '$.nama_produk',
    qty INT PATH '$.qty',
    harga DECIMAL(12,2) PATH '$.harga'
  )
) AS jt;
```

Prinsip:

- `$.items[*]` berarti ambil semua item dalam array `items`.
- `COLUMNS` mendefinisikan kolom hasil.
- Cocok untuk laporan transaksi berbasis JSON.

Lampiran B — Template Database Praktik

Lampiran ini menyediakan DDL yang dapat dipakai sepanjang buku.

B.1 Tabel mahasiswa_profile

```
CREATE TABLE mahasiswa_profile (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL UNIQUE,
  nama VARCHAR(100) NOT NULL,
  prodi VARCHAR(100) NOT NULL,
  profile JSON NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Kegunaan:

- latihan dasar JSON;
- profil mahasiswa;
- query `$.alamat.kota`;
- array `skill`;
- validasi awal.

B.2 Tabel produk

```
CREATE TABLE produk (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nama_produk VARCHAR(100) NOT NULL,
  kategori VARCHAR(50) NOT NULL,
  harga DECIMAL(12,2) NOT NULL,
  spesifikasi JSON NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Kegunaan:

- katalog produk;
- spesifikasi fleksibel;
- `JSON_SET`;
- `JSON_REMOVE`;
- pencarian berdasarkan tag.

B.3 Tabel orders_json

```
CREATE TABLE orders_json (
  order_id INT AUTO_INCREMENT PRIMARY KEY,
  customer_name VARCHAR(100) NOT NULL,
  customer_city VARCHAR(100) NOT NULL,
  order_code VARCHAR(30) NOT NULL UNIQUE,
  order_data JSON NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Kegunaan:

- transaksi berbasis JSON;
- array `items`;
- laporan dengan `JSON_TABLE`;
- agregasi total order.

B.4 Tabel api_logs

```
CREATE TABLE api_logs (
  id INT AUTO_INCREMENT PRIMARY KEY,
  source_name VARCHAR(100) NOT NULL,
  endpoint VARCHAR(200) NOT NULL,
  method VARCHAR(10) NOT NULL,
  status_code INT,
  request_payload JSON,
  response_payload JSON,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Kegunaan:

- menyimpan log API;
- audit request dan response;
- debugging integrasi;
- latihan masking data sensitif.

B.5 Tabel workflow_events

```
CREATE TABLE workflow_events (
  id INT AUTO_INCREMENT PRIMARY KEY,
  event_type VARCHAR(50) NOT NULL,
  source_name VARCHAR(100) NOT NULL,
  status VARCHAR(30) NOT NULL,
  event_data JSON,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Kegunaan:

- logging workflow n8n;
- audit trail sederhana;
- pelacakan error;
- laporan proses.

B.6 Tabel validated_students

```
CREATE TABLE validated_students (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nim VARCHAR(20) NOT NULL UNIQUE,
  nama VARCHAR(100) NOT NULL,
  email VARCHAR(150) NOT NULL UNIQUE,
  prodi VARCHAR(100) NOT NULL,
  angkatan INT NOT NULL,
  profile JSON NOT NULL,
  email_gen VARCHAR(150)
    GENERATED ALWAYS AS (profile->>'$.email') STORED,
  kota_gen VARCHAR(100)
    GENERATED ALWAYS AS (profile->>'$.alamat.kota')
STORED,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  INDEX idx_email_gen (email_gen),
  INDEX idx_kota_gen (kota_gen)
);
```

Kegunaan:

- validasi JSON Schema;
- generated column;
- optimasi query;
- proyek akhir.

Lampiran C — Contoh File JSON Latihan

Lampiran ini berisi dataset kecil untuk latihan. Data menggunakan contoh dummy.

C.1 Dataset Mahasiswa

```
[
  {
    "nim": "2026001",
    "nama": "Alya Pratama",
    "email": "alya@example.test",
    "prodi": "Informatika",
    "angkatan": 2026,
    "alamat": {
      "kota": "Tangerang Selatan",
      "provinsi": "Banten"
    },
    "skill": ["Python", "SQL", "JSON"]
  },
  {
    "nim": "2026002",
    "nama": "Bima Santoso",
    "email": "bima@example.test",
    "prodi": "Sistem Informasi",
    "angkatan": 2026,
    "alamat": {
      "kota": "Depok",
      "provinsi": "Jawa Barat"
    },
    "skill": ["API", "n8n", "Database"]
  }
]
```

C.2 Dataset Produk

```
[
  {
    "nama_produk": "Laptop Edukasi 14",
    "kategori": "laptop",
    "harga": 6500000,
    "spesifikasi": {
      "warna": "abu-abu",
      "ram_gb": 8,
      "storage_gb": 256,
      "ukuran_layar_inci": 14,
      "tag": ["edukasi", "portable", "lab"]
    }
  },
  {
    "nama_produk": "Router Lab Kampus",
    "kategori": "jaringan",
    "harga": 850000,
    "spesifikasi": {
      "warna": "putih",
      "jumlah_port": 4,
      "standar_wifi": "802.11ac",
      "tag": ["jaringan", "praktikum", "lab"]
    }
  }
]
```

C.3 Dataset Transaksi

```
[
  {
    "order_code": "ORD-2026-001",
    "customer": {
      "nama": "Alya Pratama",
      "kota": "Tangerang Selatan"
    },
    "status": "paid",
    "items": [
      {
        "sku": "BK-JSON-001",
        "nama_produk": "Buku JSON Praktis",
        "kategori": "buku",
        "qty": 2,
        "harga": 75000
      },
      {
        "sku": "USB-032",
        "nama_produk": "Flashdisk 32GB",
        "kategori": "aksesori",
        "qty": 1,
        "harga": 65000
      }
    ]
  }
]
```

C.4 Dataset Log Aplikasi

```
[
  {
    "timestamp": "2026-05-26T10:15:00+07:00",
    "app": "portal-akademik",
    "event": "login_failed",
    "user": "student01",
    "source_ip": "192.168.10.20",
    "severity": "medium"
  },
  {
    "timestamp": "2026-05-26T10:16:20+07:00",
    "app": "portal-akademik",
    "event": "login_success",
    "user": "student02",
    "source_ip": "192.168.10.21",
    "severity": "info"
  }
]
```

C.5 Dataset Response API

```
{
  "status": "success",
  "count": 2,
  "data": [
    {
      "nim": "2026001",
      "nama": "Alya Pratama",
      "prodi": "Informatika"
    },
    {
      "nim": "2026002",
      "nama": "Bima Santoso",
      "prodi": "Sistem Informasi"
    }
  ]
}
```

C.6 Dataset UMKM Sederhana

```
[
  {
    "id_umkm": "UMKM-001",
    "nama_usaha": "Kopi Serpong",
    "pemilik": "Rani",
    "kota": "Tangerang Selatan",
    "kategori": "makanan_minuman",
    "kontak": {
      "email": "kopi.serpong@example.test",
      "hp": "081200000001"
    },
    "produk": ["kopi tubruk", "kopi susu", "roti bakar"]
  },
  {
    "id_umkm": "UMKM-002",
    "nama_usaha": "Batik Depok Kreatif",
    "pemilik": "Dimas",
    "kota": "Depok",
    "kategori": "fashion",
    "kontak": {
      "email": "batik.depok@example.test",
      "hp": "081200000002"
    },
    "produk": ["kain batik", "kemeja batik"]
  }
]
```

Lampiran D — Script Python Lengkap

Lampiran ini berisi template script inti. Python dipakai untuk membaca JSON, memvalidasi, menyimpan ke MySQL, export data, transaksi, dan model SQLAlchemy.

Dokumentasi `jsonschema` menyediakan validator dan `FormatChecker` untuk validasi JSON Schema seperti email dan tanggal. (Python.org) SQLAlchemy mendukung tipe JSON untuk MySQL dan operasi JSON yang diterjemahkan ke fungsi database seperti `JSON_EXTRACT`. (SQLAlchemy)

D.1 Koneksi MySQL

```
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

try:
    conn = mysql.connector.connect(**DB_CONFIG)

    if conn.is_connected():
        print("Koneksi MySQL berhasil")

except Error as err:
    print("Koneksi gagal:", err)

finally:
    if "conn" in locals() and conn.is_connected():
        conn.close()
```

D.2 Import JSON ke MySQL

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

def load_json(filename):
    with open(filename, "r", encoding="utf-8") as file:
        return json.load(file)

def import_students(filename):
    conn = None
    cursor = None

    try:
        students = load_json(filename)

        conn = mysql.connector.connect(**DB_CONFIG)
        cursor = conn.cursor(prepared=True)

        sql = """
            INSERT INTO mahasiswa_profile
              (nim, nama, prodi, profile)
            VALUES
              (%s, %s, %s, %s)
        """

        for student in students:
            profile = {
                "email": student["email"],
                "alamat": student["alamat"],
                "skill": student["skill"]
            }

            cursor.execute(sql, (
                student["nim"],
                student["nama"],
                student["prodi"],
                json.dumps(profile, ensure_ascii=False)
            ))
```

```
            conn.commit()
            print("Import berhasil")

        except Error as err:
            if conn:
                conn.rollback()
            print("Error database:", err)

        except Exception as err:
            if conn:
                conn.rollback()
            print("Error umum:", err)

        finally:
            if cursor:
                cursor.close()
            if conn and conn.is_connected():
                conn.close()

import_students("mahasiswa.json")
```

D.3 Validasi JSON Schema

```
import json
from jsonschema import Draft202012Validator, FormatChecker

def load_json(filename):
    with open(filename, "r", encoding="utf-8") as file:
        return json.load(file)

schema = load_json("student_schema.json")
students = load_json("mahasiswa.json")

validator = Draft202012Validator(
    schema,
    format_checker=FormatChecker()
)

valid_data = []
invalid_data = []

for index, student in enumerate(students):
    errors = sorted(
        validator.iter_errors(student),
        key=lambda error: list(error.path)
    )

    if errors:
        invalid_data.append({
            "row": index + 1,
            "nim": student.get("nim"),
            "errors": [
                {
                    "path": ".".join(str(item) for item in
error.path),
                    "message": error.message
                }
                for error in errors
            ]
        })
    else:
        valid_data.append(student)

with open("valid_data.json", "w", encoding="utf-8") as file:
    json.dump(valid_data, file, ensure_ascii=False, indent=2)

with open("invalid_data.json", "w", encoding="utf-8") as file:
    json.dump(invalid_data, file, ensure_ascii=False, indent=2)

print("Valid:", len(valid_data))
print("Invalid:", len(invalid_data))
```

D.4 Export JSON dari MySQL

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

try:
    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor(dictionary=True)

    sql = """
        SELECT
          nim,
          nama,
          prodi,
          profile
        FROM mahasiswa_profile
        ORDER BY nim
    """

    cursor.execute(sql)
    rows = cursor.fetchall()

    result = []

    for row in rows:
        result.append({
            "nim": row["nim"],
            "nama": row["nama"],
            "prodi": row["prodi"],
            "profile": json.loads(row["profile"])
        })

    with open("result.json", "w", encoding="utf-8") as file:
        json.dump(result, file, ensure_ascii=False, indent=2)

    print("Export selesai")

except Error as err:
    print("Error database:", err)

finally:
    if "cursor" in locals():
        cursor.close()
    if "conn" in locals() and conn.is_connected():
        conn.close()
```

D.5 Batch Insert

```
import json
import mysql.connector
from mysql.connector import Error

DB_CONFIG = {
    "host": "localhost",
    "user": "json_user",
    "password": "GantiPasswordKuat",
    "database": "json_lab"
}

try:
    with open("mahasiswa.json", "r", encoding="utf-8") as file:
        students = json.load(file)

    rows = []

    for student in students:
        profile = {
            "email": student["email"],
            "alamat": student["alamat"],
            "skill": student["skill"]
        }

        rows.append((
            student["nim"],
            student["nama"],
            student["prodi"],
            json.dumps(profile, ensure_ascii=False)
        ))

    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor(prepared=True)

    sql = """
        INSERT INTO mahasiswa_profile
          (nim, nama, prodi, profile)
        VALUES
          (%s, %s, %s, %s)
    """

    cursor.executemany(sql, rows)
    conn.commit()

    print("Batch insert berhasil:", cursor.rowcount)

except Error as err:
    if "conn" in locals():
        conn.rollback()
    print("Error database:", err)

finally:
    if "cursor" in locals():
        cursor.close()
    if "conn" in locals() and conn.is_connected():
        conn.close()
```

D.6 Error Handling dan Transaction

```
try:
    conn = mysql.connector.connect(**DB_CONFIG)
    cursor = conn.cursor(prepared=True)

    conn.start_transaction()

    cursor.execute(
        """
        INSERT INTO workflow_events
          (event_type, source_name, status, event_data)
        VALUES
          (%s, %s, %s, %s)
        """,
        (
            "import_started",
            "python_script",
            "success",
            '{"message":"import dimulai"}'
        )
    )

    conn.commit()
    print("Transaksi sukses")

except Error as err:
    if "conn" in locals():
        conn.rollback()
    print("Transaksi dibatalkan:", err)

finally:
    if "cursor" in locals():
        cursor.close()
    if "conn" in locals() and conn.is_connected():
        conn.close()
```

D.7 SQLAlchemy Model

```
from sqlalchemy import Integer, String, JSON, DateTime, func,
create_engine
from sqlalchemy.orm import DeclarativeBase, Mapped,
mapped_column, Session

DB_URL =
"mysql+mysqlconnector://json_user:GantiPasswordKuat@localhost
/json_lab"

engine = create_engine(DB_URL, echo=True)

class Base(DeclarativeBase):
    pass

class UserProfile(Base):
    __tablename__ = "user_profiles"

    id: Mapped[int] = mapped_column(Integer,
primary_key=True, autoincrement=True)
    nim: Mapped[str] = mapped_column(String(20), unique=True,
nullable=False)
    nama: Mapped[str] = mapped_column(String(100),
nullable=False)
    profile: Mapped[dict] = mapped_column(JSON,
nullable=False)
    created_at: Mapped[str] = mapped_column(DateTime,
server_default=func.now())

Base.metadata.create_all(engine)

with Session(engine) as session:
    user = UserProfile(
        nim="2026001",
        nama="Alya Pratama",
        profile={
            "email": "alya@example.test",
            "alamat": {
                "kota": "Tangerang Selatan",
                "provinsi": "Banten"
            },
            "skill": ["Python", "SQL"]
        }
    )

    session.add(user)
    session.commit()
```

Lampiran E — Workflow n8n JSON

Lampiran ini menjelaskan pola workflow. n8n Webhook node dapat menerima data dari aplikasi atau layanan lain dan memulai workflow. (n8n Docs) n8n juga memiliki node MySQL untuk operasi seperti query SQL, insert, dan update row. (n8n Docs)

E.1 Workflow Webhook ke MySQL

Alur:

```
Webhook
  ↓
Edit Fields
  ↓
Code Validasi
  ↓
IF Valid?
  ├── True  → MySQL Insert → Respond Success
  └── False → Respond Error
```

Payload masuk:

```
{
  "nim": "2026001",
  "nama": "Alya Pratama",
  "email": "alya@example.test",
  "prodi": "Informatika",
  "angkatan": 2026,
  "profile": {
    "alamat": {
      "kota": "Tangerang Selatan",
      "provinsi": "Banten"
    },
    "skill": ["Python", "SQL"]
  }
}
```

Response sukses:

```
{
  "status": "success",
  "message": "Data berhasil disimpan",
  "nim": "2026001"
}
```

E.2 Workflow API ke MySQL

Alur:

```
Schedule Trigger / Manual Trigger
  ↓
HTTP Request
  ↓
Code Transform JSON
  ↓
MySQL Insert / Update
  ↓
Workflow Events Log
```

Kegunaan:

- mengambil data JSON dari API;
- membersihkan field;
- menyimpan ke MySQL;
- mencatat hasil proses.

E.3 Workflow Transform JSON

Alur:

```
Input JSON
  ↓
Edit Fields
  ↓
Code Node
  ↓
Output JSON bersih
```

Contoh input:

```
{
  "studentName": "Alya Pratama",
  "student_id": "2026001",
  "major": "Informatika"
}
```

Contoh output:

```
{
  "nim": "2026001",
  "nama": "Alya Pratama",
  "prodi": "Informatika"
}
```

E.4 Workflow Validasi Sederhana

Code node:

```
const item = $input.first().json;
const errors = [];

if (!item.nim) errors.push("nim wajib ada");
if (!item.nama) errors.push("nama wajib ada");
if (!item.email) errors.push("email wajib ada");
if (typeof item.angkatan !== "number") errors.push("angkatan
harus angka");

if (errors.length > 0) {
  return [{ json: { valid: false, errors, input: item } }];
}

return [{ json: { valid: true, data: item } }];
```

Catatan:

- validasi ini sederhana;
- validasi formal tetap memakai JSON Schema + Python;
- gunakan n8n untuk orkestrasi workflow.

E.5 Workflow Reporting Otomatis

Alur:

```
Schedule Trigger
  ↓
MySQL Query Report
  ↓
Code Format Result
  ↓
Write File / HTTP Response / Notification
```

Query laporan:

```
SELECT
  prodi,
  COUNT(*) AS jumlah_mahasiswa
FROM validated_students
GROUP BY prodi
ORDER BY jumlah_mahasiswa DESC;
```

Output:

```
{
  "status": "success",
  "report": "jumlah_mahasiswa_per_prodi",
  "data": [
    {
      "prodi": "Informatika",
      "jumlah_mahasiswa": 12
    }
  ]
}
```

E.6 Workflow Error Notification

Alur:

```
Workflow Error
  ↓
Code Format Error
  ↓
workflow_events Insert
  ↓
Notification
```

Payload log:

```
{
  "event_type": "workflow_error",
  "source_name": "student_registration",
  "status": "failed",
  "error": "email duplikat"
}
```

Simpan ke tabel:

```
INSERT INTO workflow_events
  (event_type, source_name, status, event_data)
VALUES
  ('workflow_error', 'student_registration', 'failed',
CAST(:event_data AS JSON));
```

Lampiran F — Checklist Keamanan dan Validasi

Lampiran ini menjadi checklist wajib sebelum proyek dipresentasikan.

F.1 Prepared Statement

Checklist:

- Semua query Python memakai parameter `%s`.
- Tidak ada input pengguna yang digabung langsung ke string SQL.
- Insert, update, dan select berbasis input memakai prepared cursor.
- Query untuk laporan tidak menerima input liar.

Contoh benar:

```
cursor.execute(
    "SELECT * FROM validated_students WHERE nim = %s",
    (nim,)
)
```

F.2 User Privilege

Checklist:

- Aplikasi tidak memakai user administratif.
- User aplikasi hanya punya hak yang diperlukan.
- Hak `DROP`, `CREATE`, dan `GRANT` tidak diberikan ke user aplikasi.
- Setiap proyek memakai user database khusus.

Contoh:

```
CREATE USER 'student_app'@'localhost'
IDENTIFIED BY 'GantiPasswordAplikasi';

GRANT SELECT, INSERT, UPDATE
ON json_lab.validated_students
TO 'student_app'@'localhost';

FLUSH PRIVILEGES;
```

F.3 JSON Validation

Checklist:

- JSON syntax divalidasi.
- JSON Schema dibuat.
- Field wajib memakai `required`.
- Email memakai `format`.
- Nilai pilihan memakai `enum`.
- Angka memakai `minimum` dan `maximum`.
- Data invalid tidak masuk MySQL.
- Laporan error validasi disimpan.

F.4 Backup

Checklist:

- Backup database dibuat sebelum demo.
- File backup disimpan di folder proyek.
- Restore pernah diuji minimal sekali.
- Backup tidak berisi data sensitif asli.
- Nama file backup memakai tanggal.

Contoh:

```
mysqldump -u root -p json_lab > backup_json_lab.sql
```

Restore:

```
mysql -u root -p json_lab < backup_json_lab.sql
```

F.5 Logging

Checklist:

- Tabel `workflow_events` tersedia.
- Event sukses dicatat.
- Event gagal dicatat.
- Error validasi dicatat.
- Error database dicatat.
- Log tidak menyimpan password.

Contoh event:

```
{
  "event_type": "validation_failed",
  "source_name": "student_import",
  "status": "failed",
  "nim": "2026003"
}
```

F.6 Masking Data Sensitif

Checklist:

- Email dimasking untuk demo publik.
- Nomor kontak dimasking.
- Credential tidak masuk dokumentasi.
- File `.env` tidak dibagikan.
- Dataset demo memakai data dummy.

Contoh masking:

```
alya@example.test → a***@example.test
081200000001 → 0812******01
```

F.7 Audit Trail Sederhana

Checklist:

- Setiap import punya catatan waktu.
- Setiap workflow punya `source_name`.
- Data invalid punya laporan.
- Perubahan penting dicatat.
- Backup dibuat setelah perubahan besar.
- Workflow n8n diekspor sebagai bukti.

Lampiran G — Rubrik Proyek Akhir Mahasiswa

Rubrik ini dipakai untuk menilai proyek akhir. Total nilai **100**.

G.1 Rubrik Penilaian

Aspek	Bobot	Kriteria Utama
Desain database	15	Skema hybrid jelas, kolom inti tepat, kolom JSON tidak liar
Kualitas struktur JSON	10	Key konsisten, nested object masuk akal, array digunakan tepat
Validasi JSON Schema	15	Schema lengkap, field wajib, tipe data, enum, format email
Keamanan	10	Prepared statement, privilege terbatas, masking, backup
Workflow n8n	15	Webhook berjalan, transformasi jelas, insert/query MySQL berhasil
Script Python	15	Import, validasi, export, error handling, transaksi
Optimasi query	8	Generated column, index, `EXPLAIN`, perbandingan sebelum/sesudah
Dokumentasi	7	README jelas, struktur folder, cara install, cara menjalankan
Presentasi	5	Singkat, jelas, menunjukkan demo dan lesson learned
Total	**100**	

G.2 Level Penilaian

A — Sangat Baik

Ciri:

- pipeline end-to-end berjalan;
- validasi kuat;
- query laporan jelas;
- keamanan dasar diterapkan;
- workflow n8n diekspor;
- dokumentasi rapi;
- mahasiswa bisa menjelaskan setiap bagian.

B — Baik

Ciri:

- fungsi utama berjalan;
- validasi ada tetapi belum lengkap;
- laporan dasar ada;
- dokumentasi cukup;
- masih ada kekurangan kecil pada error handling atau optimasi.

C — Cukup

Ciri:

- hanya sebagian pipeline berjalan;
- validasi lemah;
- dokumentasi minim;
- keamanan belum konsisten;
- query laporan masih sederhana.

D — Kurang

Ciri:

- data bisa masuk tetapi tanpa validasi;
- tidak ada schema;
- tidak ada workflow lengkap;
- tidak ada backup;
- tidak bisa menjelaskan kode dengan baik.

E — Tidak Memadai

Ciri:

- proyek tidak berjalan;
- struktur tidak jelas;
- tidak ada bukti validasi;
- tidak ada dokumentasi;
- tidak ada demo yang bisa diuji.

G.3 Checklist Pengumpulan

Mahasiswa wajib mengumpulkan:

- File SQL pembuatan tabel.
- File JSON Schema.
- Dataset JSON valid.
- Dataset JSON invalid.
- Script Python validasi.
- Script Python import.
- Script Python export.
- Workflow n8n export JSON.
- Query laporan.
- Screenshot atau log hasil uji.
- File backup database.
- README.
- Slide presentasi ringkas.

G.4 Format Presentasi 7 Slide

1. **Masalah dan tujuan proyek**
2. **Arsitektur pipeline**
3. **Desain database hybrid**
4. **Validasi JSON Schema**
5. **Workflow n8n**
6. **Query laporan dan optimasi**
7. **Security, backup, dan lesson learned**

G.5 Pertanyaan Uji Saat Presentasi

Dosen dapat bertanya:

- Mengapa field tertentu menjadi kolom biasa, bukan JSON?
- Apa risiko jika semua data dimasukkan ke satu kolom JSON?
- Bagaimana cara schema menolak email invalid?
- Apa bedanya JSON valid dan data benar?
- Di mana prepared statement dipakai?
- Apa hasil `EXPLAIN` sebelum dan sesudah index?
- Apa yang terjadi jika workflow menerima payload rusak?
- Bagaimana cara restore database dari backup?
- Bagaimana data sensitif dilindungi?
- Apa bagian proyek yang akan dikembangkan berikutnya?

Glossary

Istilah	Penjelasan
Aggregation	Proses menggabungkan banyak data menjadi ringkasan, misalnya total transaksi per kota atau jumlah mahasiswa per prodi.
AI-assisted Query	Bantuan AI untuk menyusun atau menjelaskan query SQL, tetapi hasilnya tetap harus diuji dan diverifikasi oleh manusia.
API	*Application Programming Interface*, yaitu cara aplikasi berkomunikasi dan bertukar data, biasanya dalam format JSON.
API Logs	Catatan aktivitas request dan response API, termasuk endpoint, status, payload, dan waktu kejadian.
Array	Struktur JSON berbentuk daftar nilai yang ditulis dengan tanda `[ ]`, misalnya `["Python", "SQL"]`.
Audit Trail	Jejak aktivitas sistem yang mencatat siapa, kapan, dari mana, dan apa yang terjadi pada data atau workflow.
Backup	Salinan database atau file penting untuk pemulihan ketika terjadi kerusakan, kesalahan, atau kehilangan data.
Batch Insert	Teknik memasukkan banyak data sekaligus ke database agar proses lebih efisien.
BLOB	*Binary Large Object*, tipe data untuk menyimpan data biner seperti file atau data mentah, bukan untuk struktur JSON yang perlu di-query.
Code Node	Node di n8n untuk menjalankan logic tambahan, validasi sederhana, transformasi data, atau pembuatan response.
Commit	Perintah untuk menyimpan perubahan transaksi ke database secara permanen.
Constraint	Aturan yang membatasi data, misalnya wajib ada, harus angka, harus email, atau hanya boleh memilih nilai tertentu.
CRUD	Singkatan dari *Create, Read, Update, Delete*, yaitu operasi dasar database untuk membuat, membaca, mengubah, dan menghapus data.
Cursor	Objek dalam Python untuk menjalankan query SQL dan mengambil hasil dari database.

Cursor Dictionary	Cursor yang mengembalikan hasil query dalam bentuk dictionary, sehingga kolom dapat diakses berdasarkan nama.
Data Dummy	Data contoh yang tidak merepresentasikan orang atau organisasi asli, dipakai untuk latihan, demo, dan pengujian.
Data Fleksibel	Data yang strukturnya bisa berbeda antar record, sehingga cocok disimpan dalam kolom JSON.
Data Inti	Data yang penting, sering dicari, sering dihitung, atau sering dipakai laporan, sehingga sebaiknya disimpan sebagai kolom relasional.
Database Hybrid	Desain database yang menggabungkan kolom relasional untuk data inti dan kolom JSON untuk data fleksibel.
DDL	*Data Definition Language*, perintah SQL untuk membuat atau mengubah struktur database, seperti `CREATE TABLE`.
Dialect JSON Schema	Versi aturan JSON Schema yang dipakai untuk validasi, misalnya Draft 2020-12.
Edit Fields / Set Node	Node n8n untuk memilih, mengganti, membersihkan, atau membentuk field data sebelum diproses lebih lanjut.
Engine	Komponen SQLAlchemy yang menjadi pintu utama koneksi ke database.
Enum	Aturan JSON Schema yang membatasi nilai hanya pada daftar pilihan tertentu.
Error Handling	Teknik menangani error agar program tidak berhenti secara kasar dan dapat memberi pesan yang jelas.
ETL	*Extract, Transform, Load*, yaitu proses mengambil data, mengubah atau membersihkannya, lalu menyimpannya ke database.
EXPLAIN	Perintah MySQL untuk melihat rencana eksekusi query, termasuk apakah index digunakan atau tidak.
Export JSON	Proses mengeluarkan data dari database atau aplikasi menjadi file JSON.
Field	Nama atribut dalam data, misalnya `nim`, `nama`, `email`, atau `profile`.
Foreign Key	Kolom penghubung antar tabel untuk menjaga relasi dan konsistensi data.

FormatChecker	Fitur pada pustaka `jsonschema` Python untuk mengecek format tertentu seperti email atau tanggal.
Generated Column	Kolom yang nilainya dihitung otomatis dari ekspresi lain, misalnya mengambil `$.email` dari kolom JSON.
Index	Struktur tambahan di database untuk mempercepat pencarian data.
Insert	Operasi memasukkan data baru ke database.
Item n8n	Satu unit data yang diproses dalam workflow n8n, biasanya berbentuk object JSON.
JSON	*JavaScript Object Notation*, format data ringan berbasis key-value yang umum dipakai untuk API, konfigurasi, log, dan pertukaran data.
JSON_ARRAYAGG	Fungsi MySQL untuk menggabungkan hasil query menjadi array JSON.
JSON_CONTAINS	Fungsi MySQL untuk mengecek apakah dokumen JSON mengandung nilai tertentu.
JSON_EXTRACT	Fungsi MySQL untuk mengambil nilai dari dokumen JSON berdasarkan *path*.
JSON_OBJECTAGG	Fungsi MySQL untuk membuat object JSON dari pasangan key-value hasil query.
JSON_OVERLAPS	Fungsi MySQL untuk mengecek apakah dua dokumen JSON memiliki irisan nilai.
JSON Path	Alamat untuk mengambil bagian tertentu dari JSON, misalnya `$.alamat.kota` atau `$.skill[0]`.
JSON_PRETTY	Fungsi MySQL untuk menampilkan JSON dalam format yang lebih rapi dan mudah dibaca.
JSON_REMOVE	Fungsi MySQL untuk menghapus field atau elemen tertentu dari dokumen JSON.
JSON Schema	Standar untuk mendefinisikan struktur, tipe data, field wajib, dan aturan validasi dokumen JSON.
JSON_SET	Fungsi MySQL untuk mengubah nilai JSON atau menambahkan field baru pada path tertentu.
JSON_TABLE	Fungsi MySQL untuk mengubah data JSON, terutama array, menjadi bentuk tabel agar mudah di-query dan dilaporkan.

JSON_TYPE	Fungsi MySQL untuk mengetahui tipe nilai JSON, misalnya `OBJECT`, `ARRAY`, atau `STRING`.
JSON_VALID	Fungsi MySQL untuk mengecek apakah suatu teks merupakan JSON yang valid secara sintaks.
Key	Nama field dalam object JSON, misalnya `"nama"` pada `"nama": "Alya"`.
Key-Value	Pasangan nama dan nilai dalam JSON, misalnya `"prodi": "Informatika"`.
Kolom JSON	Kolom database bertipe `JSON` yang dapat menyimpan dokumen JSON valid.
Low-Code Automation	Pendekatan membangun otomasi dengan node visual dan sedikit kode, seperti workflow n8n.
Masking Data	Teknik menyamarkan data sensitif, misalnya email atau nomor kontak, agar tidak terbuka penuh.
MEMBER OF	Operator MySQL untuk mengecek apakah suatu nilai merupakan anggota array JSON.
Metadata	Data tambahan yang menjelaskan data utama, misalnya sumber data, waktu dibuat, atau versi schema.
Model ORM	Class Python dalam SQLAlchemy yang merepresentasikan tabel database.
Multi-Valued Index	Index MySQL untuk mempercepat pencarian nilai di dalam array JSON.
MySQL Community Server	Versi MySQL yang dapat digunakan untuk praktik database relasional dan JSON.
n8n	Tool otomasi workflow open source yang memakai node untuk menghubungkan API, JSON, database, dan proses lain.
Nested Object	Object JSON di dalam object JSON lain, misalnya `alamat` yang berisi `kota` dan `provinsi`.
Node	Blok kerja dalam n8n, misalnya Webhook, Code, Edit Fields, HTTP Request, dan MySQL.
Normalisasi	Prinsip desain database untuk memisahkan data ke tabel yang lebih rapi agar tidak duplikatif dan mudah dijaga.
Object	Struktur JSON berbentuk kumpulan key-value yang ditulis dengan tanda `{ }`.

ORM	*Object Relational Mapping*, pendekatan yang memetakan tabel database menjadi object dalam bahasa pemrograman.
Payload	Isi data yang dikirim melalui API, webhook, atau request HTTP, biasanya berbentuk JSON.
Prepared Statement	Teknik menjalankan SQL dengan parameter terpisah untuk mengurangi risiko *SQL injection*.
Primary Key	Kolom utama yang menjadi identitas unik setiap baris dalam tabel.
Privilege	Hak akses user database, misalnya hanya boleh `SELECT`, `INSERT`, atau `UPDATE`.
Query	Perintah untuk mengambil, mengubah, memasukkan, atau menghapus data di database.
Query JSON	Query SQL yang membaca atau memproses data di dalam kolom JSON.
Read	Operasi membaca data dari database.
Relasional	Model database berbasis tabel, kolom, baris, relasi, dan aturan konsistensi.
Reporting	Proses membuat laporan dari data, misalnya jumlah mahasiswa per prodi atau total transaksi per kota.
Restore	Proses mengembalikan database dari file backup.
Rollback	Perintah untuk membatalkan perubahan transaksi ketika terjadi error.
Schema Versioning	Praktik memberi versi pada struktur JSON agar perubahan data dapat dilacak dan dikelola.
Session	Unit kerja dalam SQLAlchemy ORM untuk mengelola object, transaksi, commit, dan rollback.
SQL Injection	Serangan yang memanfaatkan input berbahaya untuk mengubah atau menyalahgunakan query SQL.
SQLAlchemy	Pustaka Python untuk bekerja dengan database secara lebih rapi melalui Core dan ORM.
SQLAlchemy Core	Pendekatan SQLAlchemy yang lebih dekat ke SQL tetapi ditulis sebagai ekspresi Python.
SQLAlchemy ORM	Pendekatan SQLAlchemy yang memetakan tabel menjadi class dan baris menjadi object Python.

String JSON	JSON yang masih berbentuk teks dan perlu diparse atau divalidasi sebelum diproses lebih lanjut.
Tabel Relasional	Tabel database dengan kolom tetap dan tipe data jelas.
Transaction	Sekumpulan operasi database yang diperlakukan sebagai satu unit kerja, dapat disimpan dengan `COMMIT` atau dibatalkan dengan `ROLLBACK`.
Transformasi Data	Proses mengubah bentuk data, misalnya dari field tidak konsisten menjadi struktur JSON yang rapi.
Update	Operasi mengubah data yang sudah ada di database.
User MySQL	Akun database yang digunakan aplikasi atau pengguna untuk mengakses MySQL.
Validasi	Proses memeriksa apakah data memenuhi format, struktur, tipe, dan aturan yang ditentukan.
Value	Nilai dari sebuah key dalam JSON, misalnya `"Alya"` pada `"nama": "Alya"`.
VARCHAR	Tipe data teks pendek di database, cocok untuk nama, email, kode, atau status.
Virtual Environment	Lingkungan Python terpisah untuk mengelola pustaka proyek agar tidak bercampur dengan proyek lain.
Webhook	Endpoint yang menerima data dari luar dan memicu workflow secara otomatis.
Workflow	Rangkaian node di n8n yang membentuk alur kerja otomatis dari input sampai output.

Referensi

FusionAuth Team. (2022). *Breaking down JSON Web Tokens: From pros and cons to building and revoking*. Leanpub. https://leanpub.com/json-web-tokens

FusionAuth Team. (2025). *Breaking down JSON Web Tokens: From pros and cons to building and revoking*. FusionAuth. https://fusionauth.io/resources/breaking-down-json-web-tokens.pdf

Gupta, R. (2024). *Security monitoring with Wazuh: A hands-on guide to effective enterprise security using real-life use cases in Wazuh*. Packt Publishing. https://www.packtpub.com/en-th/product/security-monitoring-with-wazuh-9781837632152

Hoppscotch. (n.d.). *Hoppscotch: Open source API development ecosystem*. https://hoppscotch.io/

Itelman, R., & Viotti, J. C. (2024). *Unifying business, data, and code: Designing data products with JSON Schema*. O'Reilly Media. https://www.oreilly.com/library/view/unifying-business-data/9781098144999/

JSON Schema. (n.d.). *JSON Schema documentation*. https://json-schema.org/

JSON Schema. (n.d.). *JSON Schema: Getting started step by step*. https://json-schema.org/learn/getting-started-step-by-step

MySQL. (n.d.). *MySQL 8.4 Reference Manual: Functions that search JSON values*. Oracle. https://dev.mysql.com/doc/refman/8.4/en/json-search-functions.html

MySQL. (n.d.). *MySQL 8.4 Reference Manual: JSON functions*. Oracle. https://dev.mysql.com/doc/refman/8.4/en/json-functions.html

MySQL. (n.d.). *MySQL 8.4 Reference Manual: JSON validation functions*. Oracle. https://dev.mysql.com/doc/refman/8.4/en/json-validation-functions.html

MySQL. (n.d.). *MySQL 9.1 Reference Manual: CREATE INDEX statement*. Oracle. https://dev.mysql.com/doc/refman/9.1/en/create-index.html

MySQL. (n.d.). *MySQL 9.1 Reference Manual: JSON data type*. Oracle. https://dev.mysql.com/doc/refman/9.1/en/json.html

MySQL. (n.d.). *MySQL 9.1 Reference Manual: JSON table functions*. Oracle. https://dev.mysql.com/doc/refman/9.1/en/json-table-functions.html

MySQL. (n.d.). *MySQL Reference Manual: mysqldump — A database backup program*. Oracle. https://dev.mysql.com/doc/refman/8.4/en/mysqldump.html

MySQL. (n.d.). *MySQL Reference Manual: Prepared statements*. Oracle. https://dev.mysql.com/doc/refman/8.4/en/sql-prepared-statements.html

MySQL. (n.d.). *MySQL Reference Manual: Secure client programming*. Oracle. https://dev.mysql.com/doc/refman/8.4/en/secure-client-programming.html

MySQL. (n.d.). *MySQL Connector/Python Developer Guide: MySQLConnection.cursor() method*. Oracle. https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlconnection-cursor.html

MySQL. (n.d.). *MySQL Connector/Python Developer Guide: MySQLCursor class*. Oracle. https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor.html

n8n. (n.d.). *Code node documentation*. https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.code/

n8n. (n.d.). *Edit Fields / Set node documentation*. https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.set/

n8n. (n.d.). *Export and import workflows*. https://docs.n8n.io/workflows/export-import/

n8n. (n.d.). *MySQL node documentation*. https://docs.n8n.io/integrations/builtin/app-nodes/n8n-nodes-base.mysql/

n8n. (n.d.). *Understanding the data structure*. https://docs.n8n.io/data/data-structure/

n8n. (n.d.). *Webhook node documentation*. https://docs.n8n.io/integrations/builtin/core-nodes/n8n-nodes-base.webhook/

Nuijten, A., & Barel, P. (2023). *Modern Oracle database programming: Level up your skill set to Oracle's latest and most powerful features in SQL, PL/SQL, and JSON*. Apress. https://www.oreilly.com/library/view/modern-oracle-database/9781484291665/

Python Software Foundation. (n.d.). *json — JSON encoder and decoder*. Python documentation. https://docs.python.org/3/library/json.html

Python Software Foundation. (n.d.). *venv — Creation of virtual environments*. Python documentation. https://docs.python.org/3/library/venv.html

python-jsonschema. (n.d.). *jsonschema documentation: Creating or extending validator classes*. https://python-jsonschema.readthedocs.io/

python-jsonschema. (n.d.). *jsonschema documentation: Validating formats*. https://python-jsonschema.readthedocs.io/en/stable/validate/

SQLAlchemy. (n.d.). *MySQL and MariaDB dialect documentation*. https://docs.sqlalchemy.org/en/latest/dialects/mysql.html

SQLAlchemy. (n.d.). *SQLAlchemy ORM documentation*. https://docs.sqlalchemy.org/en/latest/orm/

SQLAlchemy. (n.d.). *SQLAlchemy Unified Tutorial*. https://docs.sqlalchemy.org/en/latest/tutorial/

Stokes, D. (2018). *MySQL and JSON: A practical programming guide*. McGraw-Hill Education. https://www.oreilly.com/library/view/mysql-and-json/9781260135459/

Stokes, D. (2020). *MySQL & JSON: A practical programming guide* (2nd ed.). Self-published. https://www.amazon.com/MySQL-JSON-Practical-Programming-Guide/dp/057878324X

Wright, A., Andrews, H., Hutton, B., & Dennis, G. (2022). *JSON Schema validation: A vocabulary for structural validation of JSON*. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-bhutton-json-schema-validation-00

Internet Engineering Task Force. (2015). *RFC 7519: JSON Web Token (JWT)*. https://www.rfc-editor.org/rfc/rfc7519

Internet Engineering Task Force. (2017). *RFC 8259: The JavaScript Object Notation (JSON) data interchange format*. https://www.rfc-editor.org/rfc/rfc8259

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