STUDI KASUS

TOPIK/JUDUL :
Sistem Manajemen Basis Data Restoran *DineFlow*

ANALISA KEBUTUHAN USER:

DineFlow adalah sistem manajemen basis data komprehensif yang dirancang untuk operasi restoran kasual cepat. Sistem ini mengelola perjalanan pelanggan yang lengkap dari pemesanan hingga pembayaran, persiapan makanan, dan pengiriman. Dokumen ini menyajikan analisis mendetail tentang persyaratan pengguna, spesifikasi fungsional, aturan bisnis, dan model entitas dasar yang akan mendorong desain basis data.

Sistem ini mendukung empat kelompok pemangku kepentingan utama: Pelanggan (termasuk anggota program loyalitas), Kasir, Staf Dapur, dan Pelayan, dengan kemampuan pengawasan manajerial. Desain ini menekankan efisiensi operasional, akurasi pesanan, dan kepuasan pelanggan melalui program loyalitas terintegrasi dan pelacakan pesanan real-time.

1. Analisis Pemangku Kepentingan

1.1 Pemangku Kepentingan Utama

1.1.1 Pelanggan

Deskripsi: Pengguna akhir yang membeli makanan dan minuman dari restoran.

Jenis Pengguna:

- **Pelanggan Tamu:** Pelanggan pertama kali atau tidak terdaftar tanpa akun loyalitas
- **Anggota Loyalitas:** Pelanggan terdaftar dengan nomor telepon yang mendapatkan reward
- **Anggota VIP:** Pelanggan bernilai tinggi dengan 100+ bintang yang menerima manfaat premium

Kebutuhan Utama:

- Pemesanan yang cepat dan akurat
- Transparansi harga dan pemrosesan pembayaran
- Pelacakan dan penukaran reward loyalitas
- Kemampuan kustomisasi pesanan
- Waktu tunggu yang wajar (≤30 menit)

1.1.2 Kasir

Deskripsi: Staf garis depan yang bertanggung jawab atas interaksi pelanggan, entri pesanan, dan pemrosesan pembayaran.

Tanggung Jawab Utama:

- Menyambut pelanggan dan menerima pesanan
- Navigasi menu dan pemilihan item
- Dokumentasi instruksi khusus
- Pemrosesan pembayaran (tunai dan kartu)
- Pembuatan kuitansi
- Inisiasi permintaan pengembalian dana
- Pendaftaran program loyalitas

Tingkat Akses Sistem: Pengguna standar dengan hak pembuatan pesanan dan pemrosesan pembayaran

1.1.3 Staf Dapur

Deskripsi: Personel yang bertanggung jawab atas persiapan makanan dan pemenuhan pesanan.

Tanggung Jawab Utama:

- Pemantauan antrian pesanan
- Persiapan makanan mengikuti spesifikasi
- Kepatuhan instruksi khusus
- Pembaruan status pesanan (Antrian → Dalam Proses → Selesai)
- Manajemen waktu untuk memenuhi SLA 30 menit
- Jaminan kualitas

Tingkat Akses Sistem: Akses tampilan dapur dengan hak modifikasi status pesanan

1.1.4 Pelayan

Deskripsi: Staf layanan yang bertanggung jawab atas pengiriman pesanan dan manajemen meja.

Tanggung Jawab Utama:

- Pemantauan penugasan meja (maksimum 6 meja per pelayan)
- Pengiriman pesanan dari dapur ke pelanggan
- Manajemen status meja (terisi/tersedia)
- Layanan pelanggan dan eskalasi masalah

- Konfirmasi pengiriman

Tingkat Akses Sistem: Pengguna standar dengan hak manajemen meja dan pelacakan pengiriman

1.1.5 Manajer

Deskripsi: Staf pengawas dengan kemampuan administratif dan pengawasan.

Tanggung Jawab Utama:

- Persetujuan/penolakan pengembalian dana
- Manajemen item menu (tambah/edit/ketersediaan)
- Pelaporan kinerja dan analitik
- Resolusi masalah dan penanganan eskalasi
- Penegakan aturan bisnis
- Rekonsiliasi harian

Tingkat Akses Sistem: Akses administratif dengan kemampuan override

1.2 Pemangku Kepentingan Sekunder

- **Pemilik Restoran:** Pengawasan keuangan dan pengambilan keputusan strategis
- **Akuntan:** Pelaporan keuangan dan kepatuhan pajak
- **Inspektur Kesehatan:** Verifikasi kepatuhan (pertimbangan masa depan)

2. Persyaratan Fungsional

2.1 Persyaratan Pelanggan

ID	Prioritas	Deskripsi Persyaratan
FR-C-001	Kritis	Pelanggan dapat memberikan nomor telepon untuk menjadi/mengidentifikasi sebagai anggota loyalitas
FR-C-002	Kritis	Sistem menampilkan saldo bintang pelanggan saat ini ketika nomor telepon dimasukkan
FR-C-003	Kritis	Pelanggan menerima 1 bintang per pesanan (terlepas dari nilai pesanan)
FR-C-004	Kritis	Pelanggan menerima diskon otomatis: 10 bintang=5% off, 25bintang=10% off, 50bintang=15% off, 100bintang=20% off

FR-C-005	Kritis	Bintang dikurangi saat diskon diterapkan
FR-C-006	Tinggi	Pelanggan dapat melihat riwayat pesanan (20 pesanan terakhir) saat nomor telepon dimasukkan
FR-C-007	Kritis	Pelanggan menerima kuitansi cetak dengan nomor pesanan, item, harga, metode pembayaran, timestamp, estimasi waktu tunggu, nomor meja (jika dine-in)
FR-C-008	Tinggi	Pelanggan dapat meminta modifikasi pesanan dalam 5 menit setelah pembuatan pesanan DAN hanya jika status pesanan = 'ANTRIAN'
FR-C-009	Tinggi	Pelanggan dapat menentukan instruksi khusus per item menu (maks 200 karakter)
FR-C-010	Sedang	Sistem melacak preferensi pelanggan untuk referensi masa depan
FR-C-011	Kritis	Pelanggan dapat memilih dine-in atau takeout saat pemesanan

2.2 Persyaratan Kasir

ID	Prioritas	Deskripsi Persyaratan
FR-CA-001	Kritis	Kasir harus login dengan ID Kasir unik di awal shift
FR-CA-002	Kritis	Kasir dapat menelusuri item menu berdasarkan kategori (Appetizer, Main, Dessert, Beverage)
FR-CA-003	Kritis	Kasir dapat menambahkan item ke pesanan dengan jumlah (1-99)
FR-CA-004	Kritis	Kasir dapat menghapus item dari pesanan sebelum pembayaran
FR-CA-005	Tinggi	Kasir dapat menambahkan instruksi khusus ke item individu
FR-CA-006	Kritis	Sistem menampilkan total pesanan real-time termasuk pajak (tarif pajak 10%)

FR-CA-007	Kritis	Kasir dapat menerapkan diskon loyalitas jika pelanggan memenuhi syarat
FR-CA-008	Kritis	Kasir dapat memproses pembayaran melalui Tunai, Kartu Kredit, atau Kartu Debit
FR-CA-009	Kritis	HANYA SATU metode pembayaran per pesanan (tidak ada pembayaran terpisah)
FR-CA-010	Kritis	Sistem menghasilkan nomor pesanan secara berurutan per hari (format: YYYYMMDD-####)
FR-CA-011	Kritis	Kasir dapat menugaskan nomor meja untuk pesanan dine-in
FR-CA-012	Tinggi	Kasir dapat memulai permintaan pengembalian dana (memerlukan persetujuan manajer)
FR-CA-013	Sedang	Kasir dapat mencetak ulang kuitansi jika pelanggan meminta
FR-CA-014	Tinggi	Kasir logout di akhir shift
FR-CA-015	Sedang	Sistem melacak semua pesanan berdasarkan ID Kasir untuk tinjauan kinerja
FR-CA-016	Tinggi	Kasir dapat mencari pelanggan berdasarkan nomor telepon untuk mengambil info loyalitas
FR-CA-017	Sedang	Kasir dapat melihat panjang antrian saat ini dan estimasi waktu tunggu
FR-CA-018	Tinggi	Kasir mengkonfirmasi instruksi khusus secara verbal dengan pelanggan sebelum pembayaran
FR-CA-019	Tinggi	Pembayaran terpisah ditangani dengan membuat pesanan terpisah dengan tautan

2.3 Persyaratan Staf Dapur

ID	Prioritas	Deskripsi Persyaratan
FR-K-001	Kritis	Tampilan dapur menunjukkan pesanan dalam antrian FIFO (First In, First Out)

FR-K-002	Kritis	Koki dapat melihat detail pesanan: nomor pesanan, item, jumlah, instruksi khusus, waktu pesanan, nomor meja
FR-K-003	Kritis	Koki dapat menandai status pesanan: ANTRIAN → DALAM_PROSES → SELESAI
FR-K-004	Tinggi	Koki dapat melihat estimasi waktu persiapan per item
FR-K-005	Kritis	Sistem memberi peringatan kepada koki ketika pesanan melebihi 25 menit (peringatan) dan 30 menit (kritis)
FR-K-006	Kritis	Koki dapat memberi tahu pelayan ketika pesanan SELESAI
FR-K-007	Sedang	Sistem menampilkan estimasi total waktu persiapan untuk seluruh pesanan
FR-K-008	Tinggi	Tampilan dapur memberi kode warna pesanan berdasarkan prioritas: Hijau (<15min), Kuning (15-25min), Merah (>25min)
FR-K-009	Rendah	Koki dapat menambahkan catatan ke pesanan

2.4 Persyaratan Pelayan

ID	Prioritas	Deskripsi Persyaratan
FR-W-001	Kritis	Pelayan login dengan ID Pelayan unik
FR-W-002	Kritis	Pelayan dapat melihat meja yang ditugaskan (maksimum 6 meja per pelayan)
FR-W-003	Kritis	Pelayan menerima notifikasi ketika pesanan untuk meja mereka SELESAI
FR-W-004	Kritis	Pelayan dapat menandai pesanan sebagai TERKIRIM ketika makanan dibawa ke meja
FR-W-005	Kritis	Pelayan dapat menandai meja sebagai TERSEDIA ketika pelanggan pergi
FR-W-006	Tinggi	Pelayan dapat menangani keluhan pelanggan dan meningkatkan ke manajer

FR-W-007	Sedang	Pelayan dapat melihat status loyalitas pelanggan untuk layanan personal
FR-W-008	Rendah	Sistem melacak rata-rata waktu pengiriman per pelayan
FR-W-009	Rendah	Pelayan dapat meminta notifikasi pembersihan meja

2.5 Persyaratan Manajer

ID	Prioritas	Deskripsi Persyaratan
FR-M-001	Kritis	Manajer dapat menyetujui/menolak permintaan pengembalian dana
FR-M-002	Kritis	Manajer dapat melihat laporan penjualan harian: total pesanan, total pendapatan, nilai pesanan rata-rata
FR-M-003	Tinggi	Manajer dapat melihat kinerja kasir: pesanan yang diproses, waktu pemrosesan rata-rata
FR-M-004	Tinggi	Manajer dapat melihat kinerja dapur: waktu persiapan rata-rata, pesanan yang melebihi 30 menit
FR-M-005	Kritis	Manajer dapat menambah/edit/menonaktifkan item menu
FR-M-006	Kritis	Manajer dapat mengatur ketersediaan item menu (tandai sebagai tidak tersedia sementara)
FR-M-007	Tinggi	Manajer dapat melihat rincian penjualan per jam untuk mengidentifikasi jam sibuk
FR-M-008	Sedang	Manajer dapat melihat statistik loyalitas pelanggan: total anggota, distribusi bintang
FR-M-009	Sedang	Manajer dapat menghasilkan laporan pendapatan mingguan/bulanan
FR-M-010	Sedang	Manajer dapat melihat item menu paling populer
FR-M-011	Tinggi	Manajer dapat mengganti aturan sistem (misalnya, aplikasi diskon manual)

FR-M-012	Tinggi	Manajer dapat melihat semua pesanan aktif di semua status

3. Persyaratan Non-Fungsional

3.1 Persyaratan Kinerja

ID	Persyaratan	Metrik Target
NFR-P-001	Waktu respons pembuatan pesanan	≤ 3 detik
NFR-P-002	Kapasitas pesanan bersamaan	50 pesanan selama jam sibuk
NFR-P-003	Tingkat refresh tampilan dapur	≤ 1 detik setelah pembaruan status
NFR-P-004	Waktu pencetakan kuitansi	≤ 2 detik setelah pembayaran
NFR-P-005	Waktu respons query basis data	≤ 500ms untuk persentil ke-95
NFR-P-006	Waktu otorisasi pembayaran	≤ 5 detik

3.2 Persyaratan Keandalan

ID	Persyaratan	Metrik Target
NFR-R-001	Uptime sistem selama jam kerja	99.5% (10 AM - 10 PM)
NFR-R-002	Frekuensi backup basis data	Harian pada 2:00 AM
NFR-R-003	Integritas transaksi	100% kepatuhan ACID
NFR-R-004	Toleransi kehilangan data	Toleransi nol (RPO = 0)
NFR-R-005	Mean Time To Recovery (MTTR)	≤ 30 menit

3.3 Persyaratan Keamanan

ID	Persyaratan	Implementasi
NFR-S-001	Autentikasi staf	Kredensial unik per karyawan
NFR-S-002	Perlindungan data pembayaran	Tidak ada penyimpanan nomor kartu lengkap (kepatuhan PCI-DSS)
NFR-S-003	Privasi data pelanggan	Nomor telepon di-hash dalam basis data
NFR-S-004	Logging aksi manajer	Semua tindakan administratif dicatat dengan timestamp dan ID pengguna
NFR-S-005	Manajemen sesi	Auto-logout setelah 30 menit tidak aktif
NFR-S-006	Enkripsi data	TLS 1.3 untuk data dalam transit

3.4 Persyaratan Kegunaan

ID	Persyaratan	Metrik Target
NFR-U-001	Efisiensi entri pesanan kasir	≤ 5 klik untuk menyelesaikan pesanan
NFR-U-002	Keterbacaan tampilan dapur	Terbaca dari jarak 10 kaki
NFR-U-003	Kejelasan pesan error	Bahasa sederhana, tanpa jargon teknis
NFR-U-004	Waktu pelatihan untuk staf baru	≤ 2 jam untuk kemahiran dasar
NFR-U-005	Kompatibilitas layar sentuh	Semua antarmuka dioptimalkan untuk sentuhan

3.5 Persyaratan Skalabilitas

ID	Persyaratan	Kapasitas Target

NFR-SC-001	Volume pesanan bulanan	Mendukung pertumbuhan hingga 10.000 pesanan/bulan
NFR-SC-002	Kapasitas item menu	Mendukung hingga 100 item menu
NFR-SC-003	Basis data pelanggan	Mendukung 50.000 anggota loyalitas
NFR-SC-004	Retensi data historis	3 tahun riwayat pesanan

4. Aturan Bisnis

4.1 Aturan Manajemen Pesanan

ID Aturan	Deskripsi Aturan
BR-001	Nomor pesanan berurutan per hari, format: YYYYMMDD-#### (mis., 20241208-0001)
BR-002	Pesanan tidak dapat dibatalkan setelah pembayaran; hanya pengembalian dana yang diizinkan
BR-003	Modifikasi pesanan hanya diizinkan jika status = 'ANTRIAN' dan dalam 5 menit setelah pembuatan
BR-004	Maksimum 30 menit dari pembuatan pesanan hingga penyelesaian (Service Level Agreement)
BR-005	Pesanan yang melebihi 30 menit memicu kelayakan pengembalian dana otomatis
BR-006	Nilai pesanan minimum: $5.00
BR-007	Item maksimum per pesanan: 50
BR-008	Progresi status pesanan harus mengikuti: PENDING → ANTRIAN → DALAM_PROSES → SELESAI → TERKIRIM
BR-009	Pesanan yang dibatalkan harus memiliki status = 'ANTRIAN' dan dalam 5 menit setelah pembuatan

4.2 Aturan Program Loyalitas

ID Aturan	Deskripsi Aturan
BR-010	1 bintang diperoleh per pesanan yang diselesaikan (bukan per jumlah dolar)
BR-011	Bintang tidak pernah kedaluwarsa
BR-012	Tingkat diskon: 10bintang=5% off, 25bintang=10% off, 50bintang=15% off, 100bintang=20% off
BR-013	Hanya satu diskon per pesanan
BR-014	Bintang dikurangi segera saat diskon diterapkan
BR-015	Nomor telepon adalah pengenal unik untuk anggota loyalitas (10 digit)
BR-016	Riwayat pesanan pelanggan disimpan selama 1 tahun
BR-017	Diskon diterapkan ke subtotal sebelum perhitungan pajak
BR-018	Pesanan yang dikembalikan tidak mendapat bintang; bintang yang diperoleh dihapus jika pesanan dikembalikan

4.3 Aturan Pembayaran

ID Aturan	Deskripsi Aturan
BR-019	Pembayaran segera (tidak ada opsi "bayar nanti")
BR-020	Satu metode pembayaran per pesanan (tidak ada pembayaran terpisah antar metode)
BR-021	Pembayaran terpisah antar pelanggan memerlukan pesanan terpisah dengan flag tautan opsional
BR-022	Tarif pajak: 10% diterapkan ke subtotal setelah diskon
BR-023	Pembayaran tunai memerlukan perhitungan kembalian jika jumlah yang diberikan melebihi total
BR-024	Kode otorisasi pembayaran kartu harus disimpan untuk rekonsiliasi
BR-025	Empat digit terakhir kartu disimpan; nomor kartu lengkap TIDAK PERNAH disimpan

BR-026	Pembayaran gagal harus memungkinkan percobaan ulang dengan metode pembayaran alternatif

4.4 Aturan Pengembalian Dana

ID Aturan	Deskripsi Aturan
BR-027	Pengembalian dana diizinkan untuk: waktu tunggu berlebihan (>30min), pesanan salah, masalah kualitas
BR-028	Semua pengembalian dana memerlukan persetujuan manajer
BR-029	Pengembalian dana harus dikeluarkan ke metode pembayaran asli
BR-030	Pesanan yang dikembalikan tidak mendapat bintang; bintang dihapus jika sudah dikreditkan
BR-031	Pengembalian dana parsial tidak diizinkan; hanya pengembalian dana pesanan penuh
BR-032	Alasan pengembalian dana harus didokumentasikan (minimum 10 karakter)
BR-033	Pengembalian dana diproses dalam 3-5 hari kerja untuk pembayaran kartu

4.5 Aturan Manajemen Meja

ID Aturan	Deskripsi Aturan
BR-034	Restoran memiliki 20 meja bernomor 1-20
BR-035	Satu pesanan per meja pada satu waktu
BR-036	Meja tetap terisi sampai pelayan menandai sebagai TERSEDIA
BR-037	Setiap pelayan ditugaskan maksimum 6 meja per shift
BR-038	Penugasan meja otomatis berdasarkan ketersediaan (nomor terendah yang tersedia)
BR-039	Pesanan takeout tidak memiliki penugasan meja

4.6 Aturan Operasi Dapur

ID Aturan	Deskripsi Aturan
BR-040	Pesanan disiapkan dalam urutan FIFO (First In, First Out)
BR-041	Koki tidak dapat melewati pesanan dalam antrian tanpa override manajer
BR-042	Instruksi khusus harus ditinjau sebelum menandai pesanan DALAM_PROSES
BR-043	Progresi status pesanan tidak dapat dibalik (tidak ada DALAM_PROSES → ANTRIAN)
BR-044	Koki harus memperbarui status dalam 2 menit setelah perubahan status aktual

4.7 Aturan Menu

ID Aturan	Deskripsi Aturan
BR-045	Item menu dikategorikan sebagai: Appetizer, Main, Dessert, Beverage
BR-046	Item menu memiliki estimasi waktu persiapan (5-20 menit)
BR-047	Item menu dapat ditandai tidak tersedia (sementara) tanpa penghapusan
BR-048	Perubahan harga memerlukan otorisasi manajer
BR-049	Item menu yang dihapus disimpan dalam basis data untuk integritas pesanan historis
BR-050	Nama item menu harus unik dalam kategori

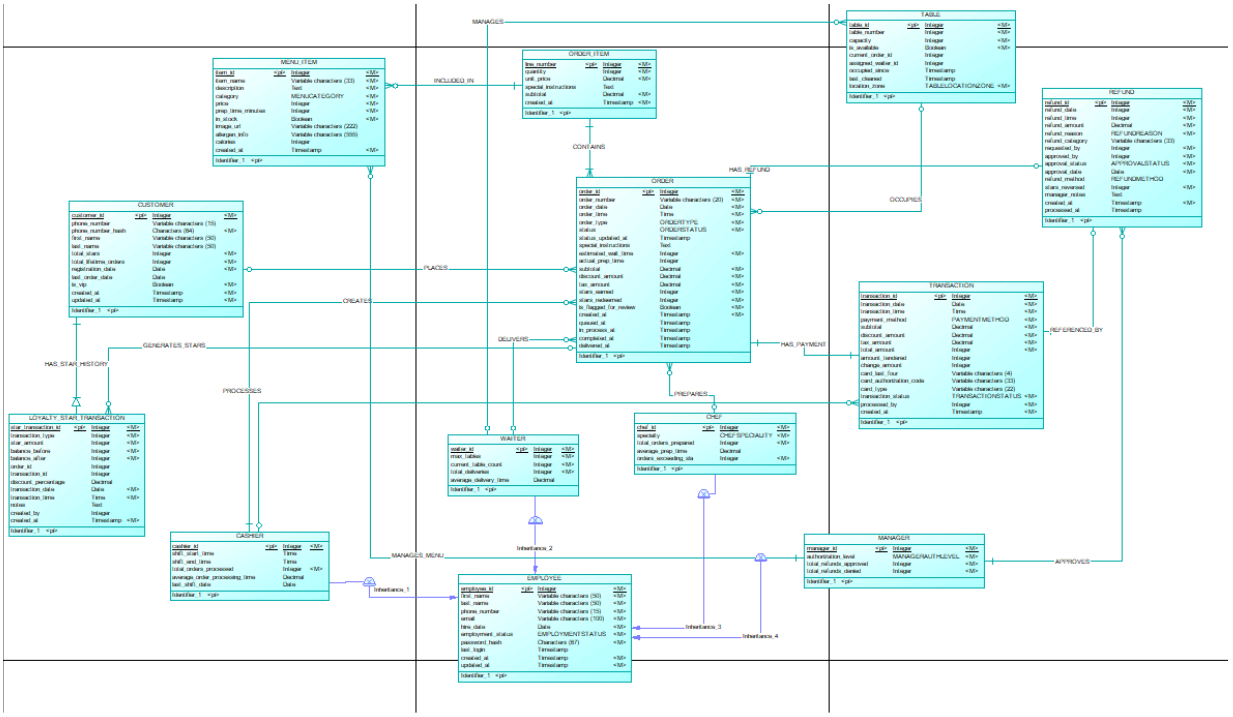
4.8 Aturan Staf

ID Aturan	Deskripsi Aturan
BR-051	Setiap anggota staf memiliki ID karyawan unik
BR-052	Peran staf terpisah (satu karyawan memiliki satu peran utama)
BR-053	Kasir dan pelayan bekerja dalam shift (maksimum 8 jam)

BR-054	Satu manajer bertugas setiap saat selama jam kerja
BR-055	Staf tidak dapat menghapus atau memodifikasi data historis (pesanan, transaksi)
BR-056	Kredensial staf kedaluwarsa setelah 90 hari (perubahan password diperlukan)

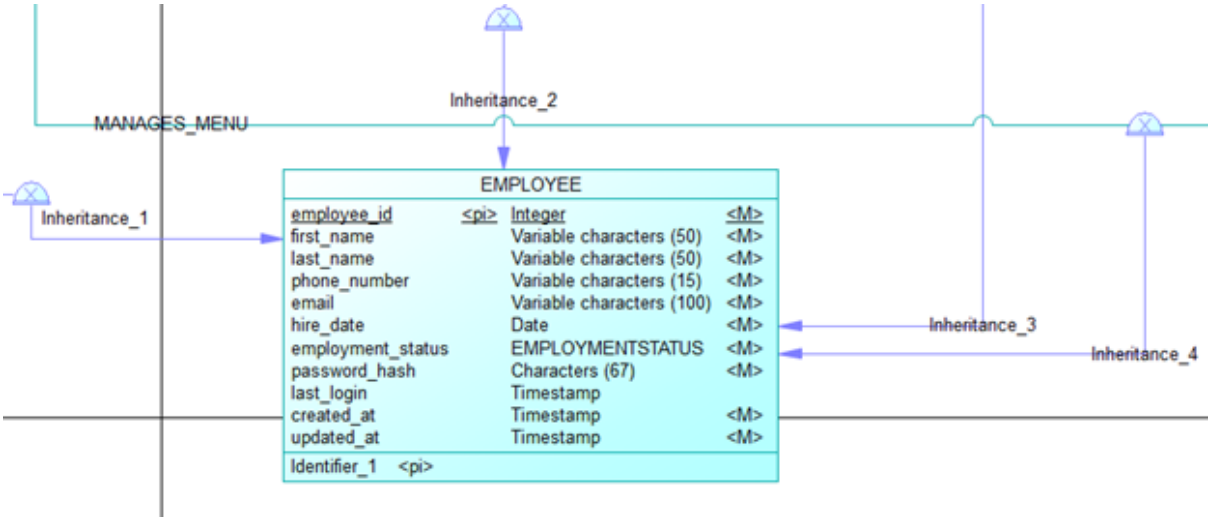
DESAIN DATABASE

DESAIN ER :



TRANSFORMASI ER to RDBMS :

1. EMPLOYEE (Supertype)

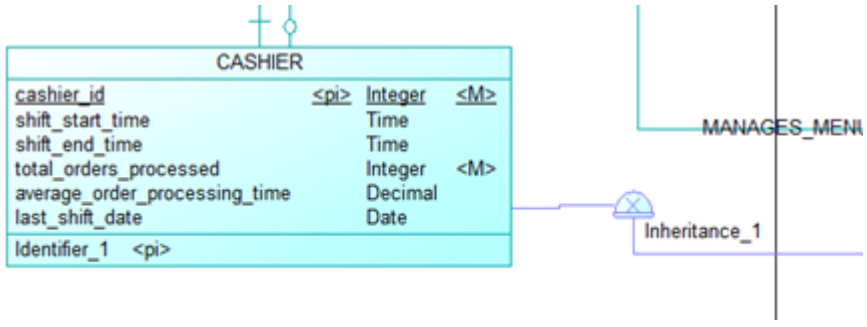


Hasil Transformasi:

EMPLOYEE (employee_id, first_name, last_name, phone_number, email, hire_date, employment_status, employee_role, password_hash, last_login, created_at, updated_at)

Primary Key: employee_id
Unique Keys: phone_number, email
Foreign Keys: None

2. CASHIER (Subtype)

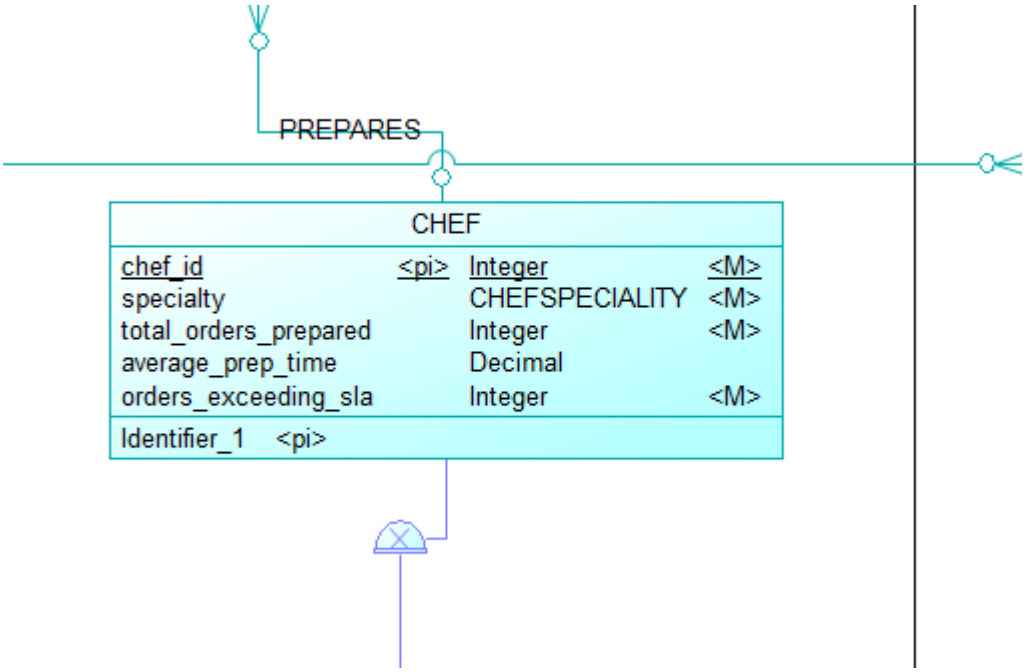


Hasil Transformasi:

CASHIER (cashier_id, shift_start_time, shift_end_time, total_orders_processed, average_order_processing_time, last_shift_date)

Primary Key: cashier_id
Foreign Keys: cashier_id → EMPLOYEE(employee_id)

3. CHEF (Subtype)

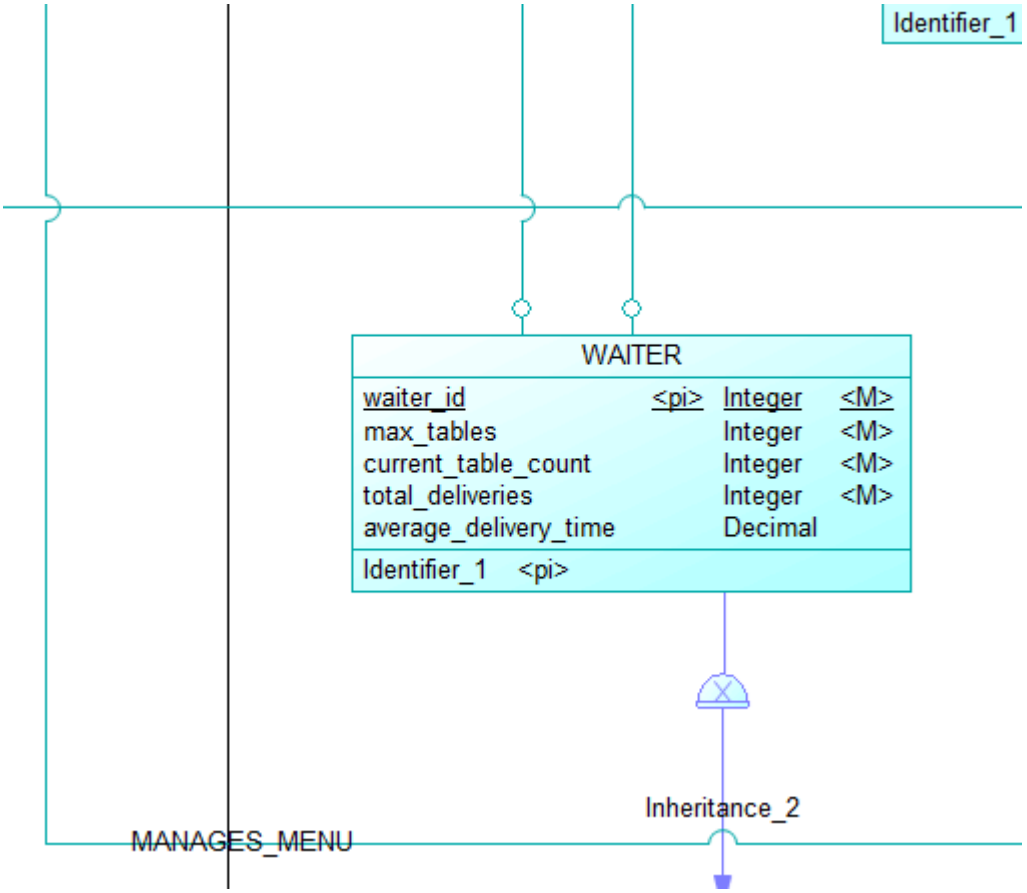


Hasil Transformasi:

CHEF (chef_id, specialty, total_orders_prepared, average_prep_time, orders_exceeding_sla)

Primary Key: chef_id
Foreign Keys: chef_id → EMPLOYEE(employee_id)

4. WAITER (Subtype)

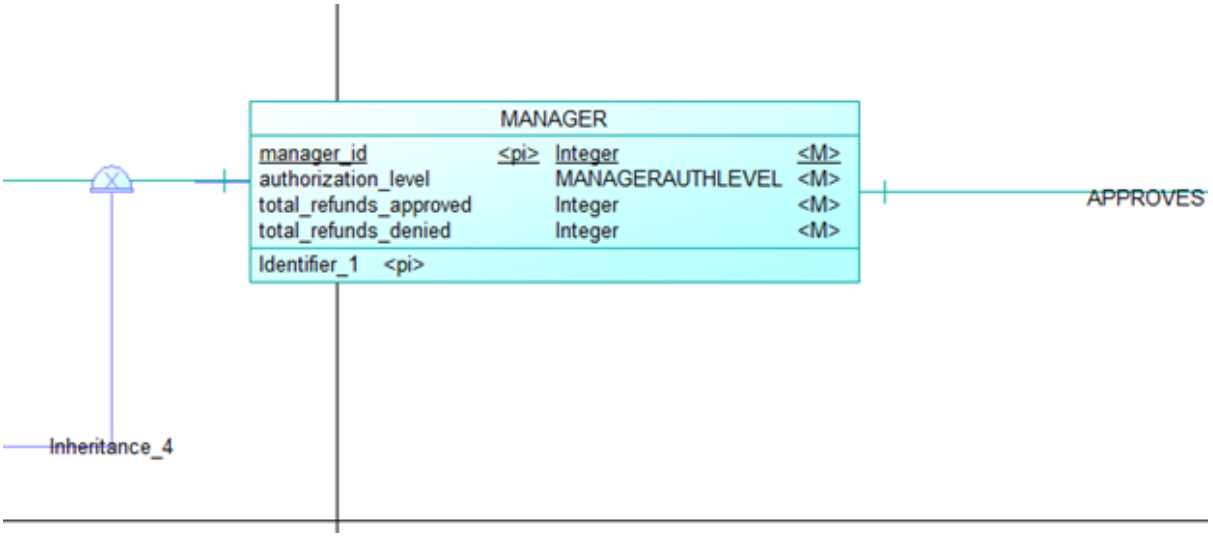


Hasil Transformasi:

WAITER (waiter_id, max_tables, current_table_count, total_deliveries, average_delivery_time)

Primary Key: waiter_id
Foreign Keys: waiter_id → EMPLOYEE(employee_id)

5. MANAGER (Subtype)

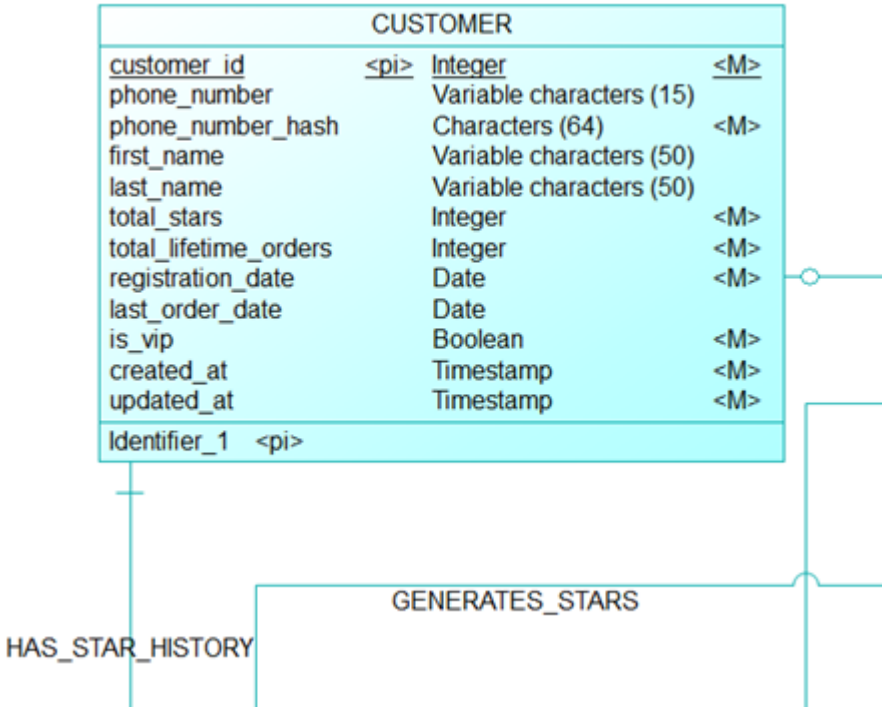


Hasil Transformasi:

MANAGER (manager_id, authorization_level, total_refunds_approved,
total_refunds_denied)

Primary Key: manager_id
Foreign Keys: manager_id → EMPLOYEE(employee_id)

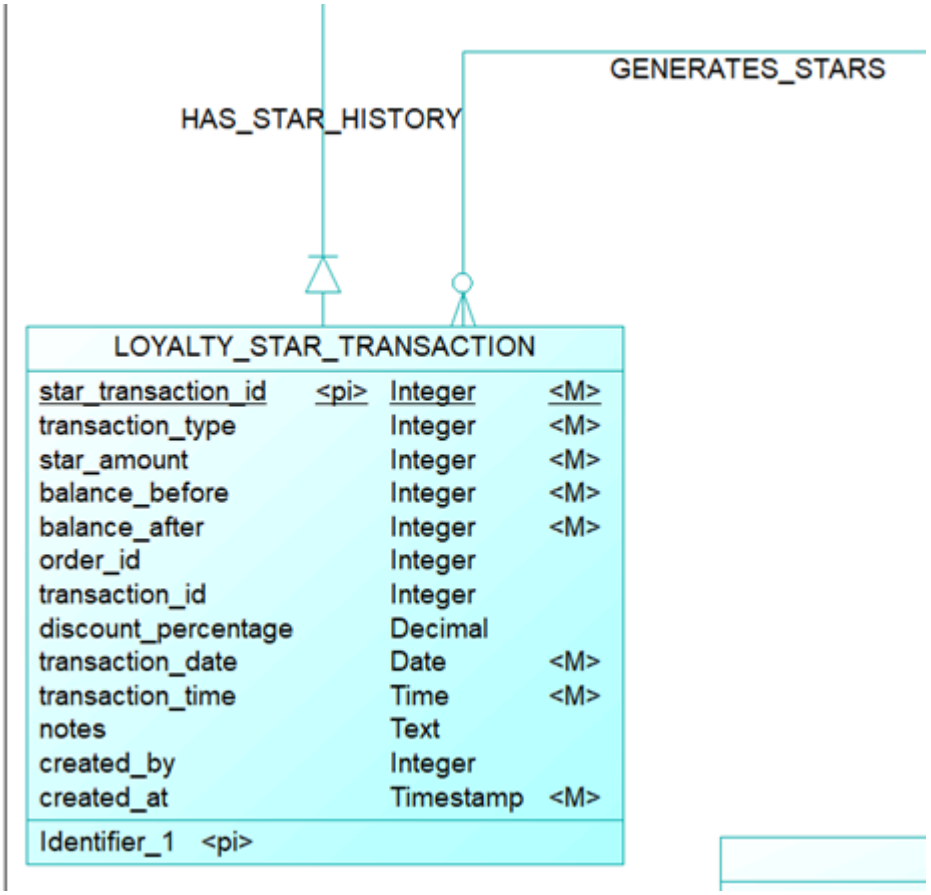
6. CUSTOMER



Hasil Transformasi:

CUSTOMER (customer_id, phone_number, phone_number_hash, first_name, last_name,
total_stars, total_lifetime_orders, registration_date, last_order_date, is_vip, created_at, updated_at)

7. LOYALTY_STAR_TRANSACTION (Weak Entity)



Hasil Transformasi:

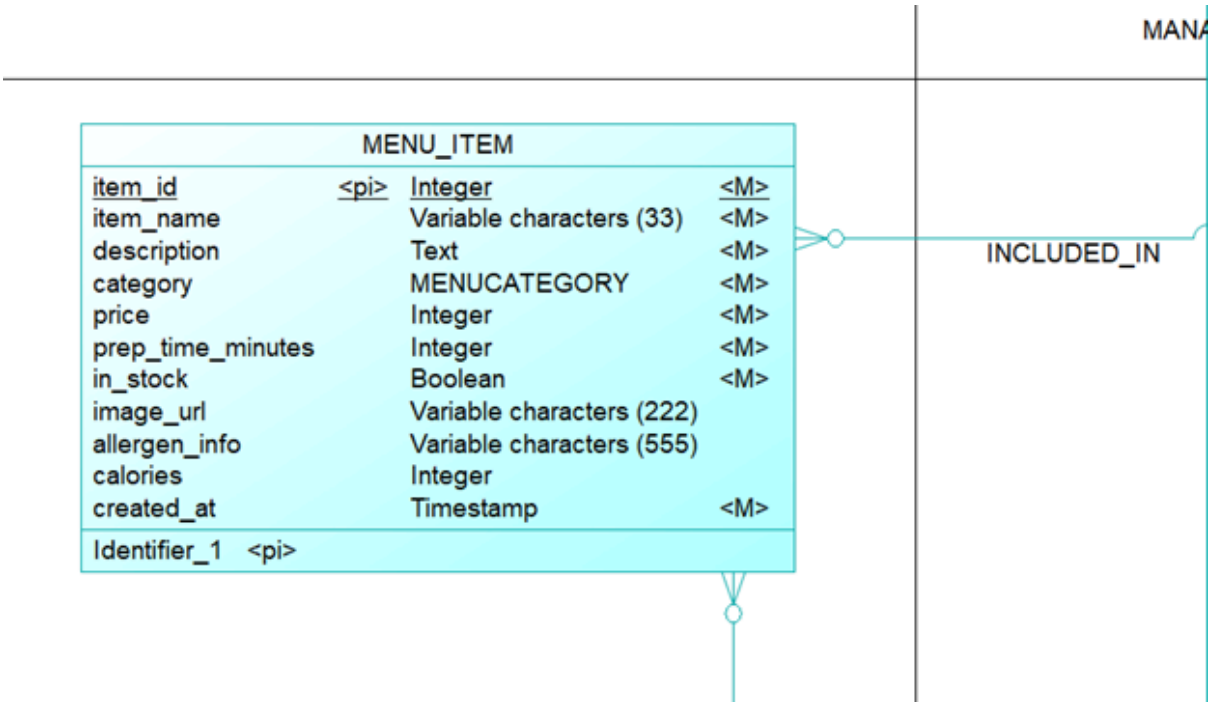
LOYALTY_STAR_TRANSACTION (star_transaction_id, customer_id, transaction_type,
star_amount, balance_before, balance_after, order_id, transaction_id, refund_id, discount_percentage, transaction_date, transaction_time, notes, created_by, created_at)

Primary Key: (star_transaction_id, customer_id) - Composite
Foreign Keys:

- customer_id → CUSTOMER(customer_id) - Identifying relationship
- order_id → ORDER(order_id)

- transaction_id → TRANSACTION(transaction_id)
- refund_id → REFUND(refund_id)
- created_by → EMPLOYEE(employee_id)

8. MENU_ITEM

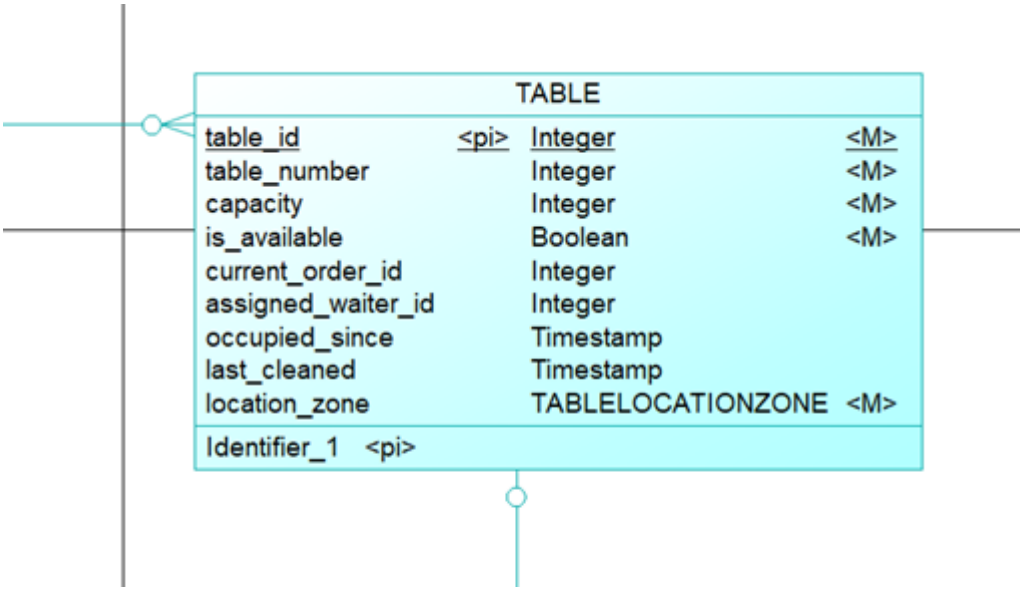


Hasil Transformasi:

MENU_ITEM (item_id, item_name, description, category, price, cost,
prep_time_minutes, is_available, is_active, image_url,
allergen_info, calories, created_at, updated_at, created_by)

Primary Key: item_id
Unique Constraint: (item_name, category) - unique within category
Foreign Keys: created_by → MANAGER(manager_id)

9. RESTAURANT_TABLE



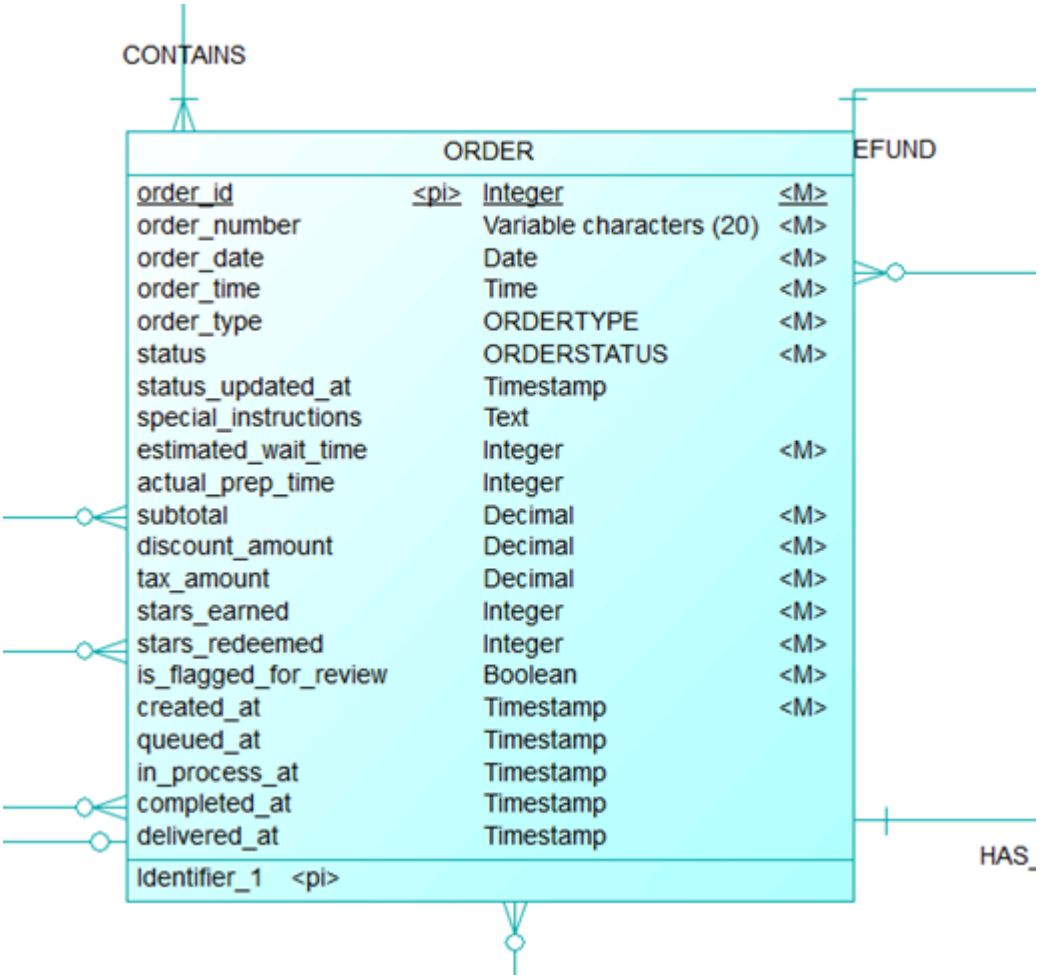
Hasil Transformasi:

RESTAURANT_TABLE (table_id, table_number, capacity, is_available,
current_order_id, assigned_waiter_id, occupied_since,
last_cleaned, location_zone, is_active)

Primary Key: table_id
Unique Key: table_number
Foreign Keys:

- assigned_waiter_id → WAITER(waiter_id)
- current_order_id → ORDER(order_id) - circular reference

10. CUSTOMER_ORDER



Hasil Transformasi:

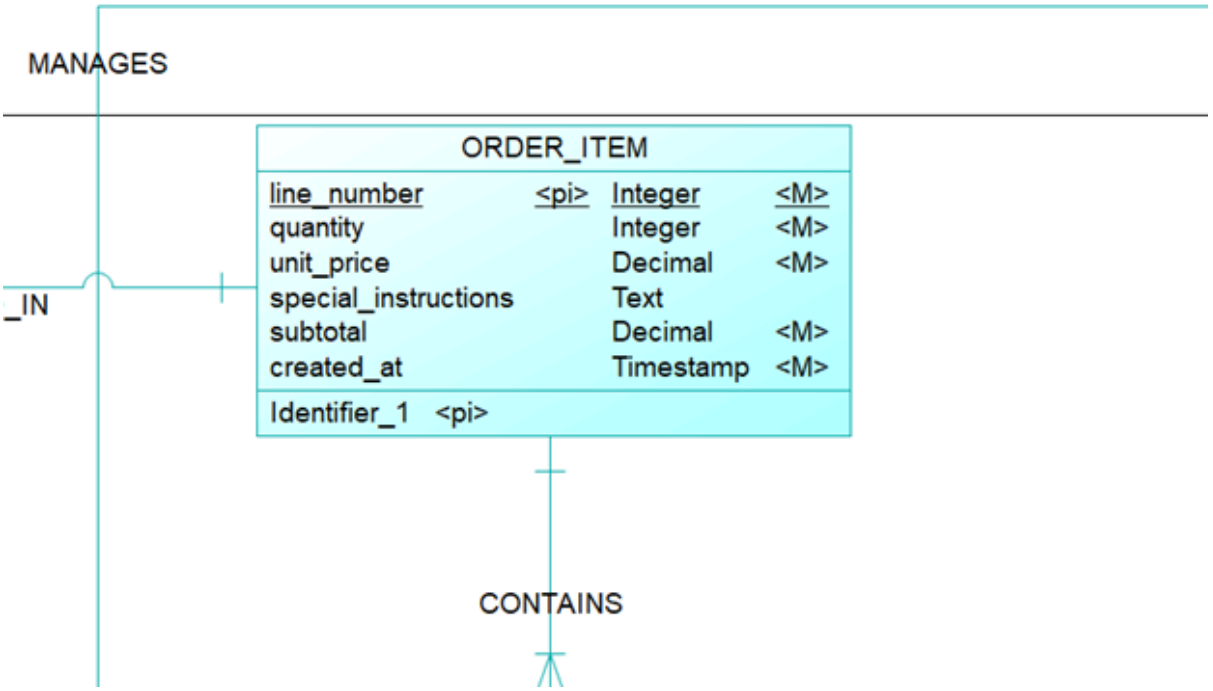
CUSTOMER_ORDER (order_id, order_number, customer_id, cashier_id, chef_id, waiter_id, table_id, order_date, order_time, order_type, status, status_updated_at, special_instructions, estimated_wait_time, actual_prep_time, subtotal, discount_amount, tax_amount, total_amount, stars_earned, stars_redeemed, is_flagged_for_review, created_at, queued_at, in_process_at, completed_at, delivered_at, related_order_id)

Primary Key: order_id
Unique Key: order_number
Foreign Keys:

- customer_id → CUSTOMER(customer_id)
- cashier_id → CASHIER(cashier_id)
- chef_id → CHEF(chef_id)
- waiter_id → WAITER(waiter_id)
- table_id → TABLE(table_id)

- related_order_id → ORDER(order_id) - self-reference

11. ORDER_ITEM (Associative Entity)



Hasil Transformasi:

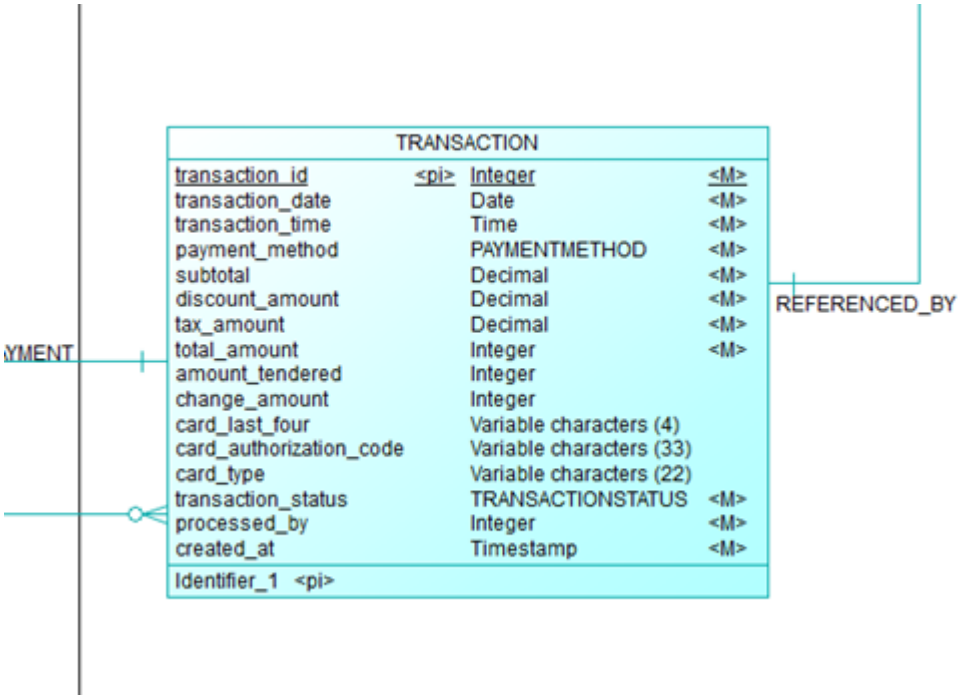
ORDER_ITEM (order_id, item_id, line_number, quantity, unit_price,
special_instructions, subtotal, created_at)

EER Concept: Associative entity for M:N relationship between ORDER and MENU_ITEM

Primary Key: (order_id, item_id, line_number) - Composite
Foreign Keys:

- order_id → ORDER(order_id)
- item_id → MENU_ITEM(item_id)

12. PAYMENT_TRANSACTION



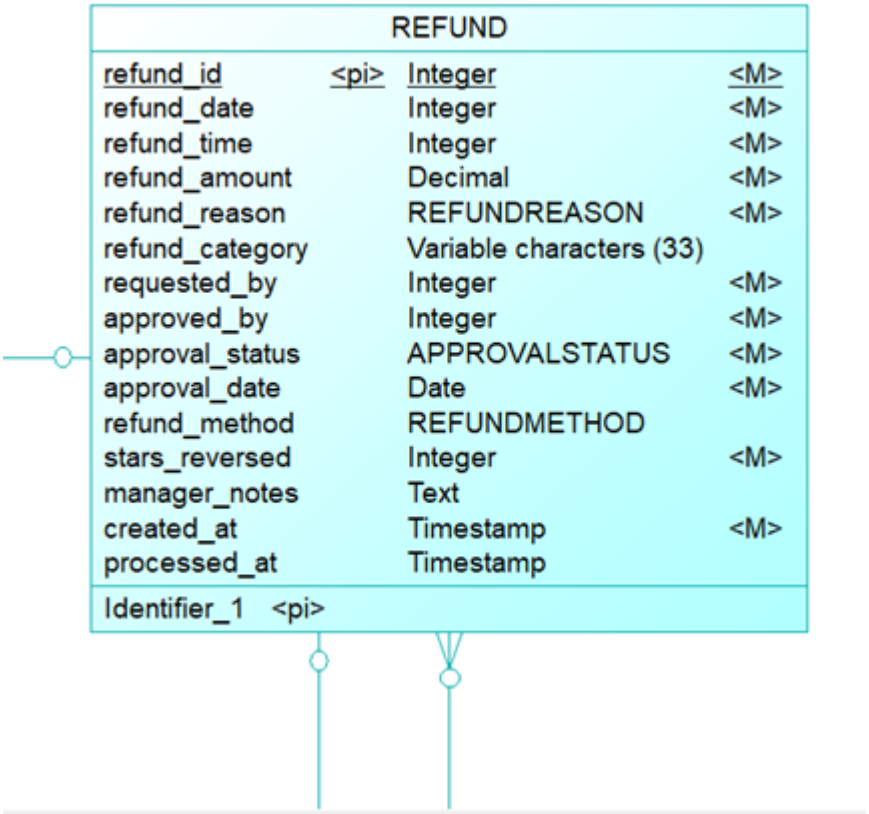
Hasil Transformasi:

PAYMENT_TRANSACTION (transaction_id, order_id, transaction_date, transaction_time,
payment_method, subtotal, discount_amount, tax_amount, total_amount, amount_tendered, change_amount, card_last_four, card_authorization_code, card_type, transaction_status, processed_by, created_at)

Primary Key: transaction_id
Unique Key: order_id (one transaction per order)
Foreign Keys:

- order_id → ORDER(order_id)
- processed_by → CASHIER(cashier_id)

13. REFUND



Hasil Transformasi:

REFUND (refund_id, order_id, transaction_id, refund_date, refund_time, refund_amount, refund_reason, reason_category, requested_by, approved_by, approval_status, approval_date, refund_method, stars_reversed, manager_notes, created_at, processed_at)

Primary Key: refund_id
Unique Key: order_id (one refund per order)
Foreign Keys:

- order_id → ORDER(order_id)
- transaction_id → TRANSACTION(transaction_id)
- requested_by → CASHIER(cashier_id)
- approved_by → MANAGER(manager_id)

HASIL TABEL :

1. EMPLOYEE (Supertype)

Hasil Transformasi:

EMPLOYEE (employee_id, first_name, last_name, phone_number, email, hire_date, employment_status, employee_role, password_hash, last_login, created_at, updated_at)

Primary Key: employee_id
Unique Keys: phone_number, email
Foreign Keys: None

Hasil Tabel:

Field	Data Type	Constraint	Description
employee_id	INT	PK, AUTO_INCREMENT	Unique employee identifier
first_name	VARCHAR(50)	NOT NULL	Employee first name
last_name	VARCHAR(50)	NOT NULL	Employee last name
phone_number	VARCHAR(15)	NOT NULL, UNIQUE	Contact phone (10-15 digits)
email	VARCHAR(100)	NOT NULL, UNIQUE	Email address (valid format)
hire_date	DATE	NOT NULL	Date employee was hired
employment_status	ENUM	NOT NULL, DEFAULT 'ACTIVE'	ACTIVE/ON_LEAVE/TERMINATED
employee_role	ENUM	NOT NULL	CASHIER/CHEF/WAITER/MANAGER
password_hash	CHAR(64)	NOT NULL	SHA-256 hashed password
last_login	TIMESTAMP	NULL	Last system login timestamp

created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Record creation timestamp
updated_at	TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP	Last update timestamp

2. CASHIER (Subtype)

Hasil Transformasi:

CASHIER (cashier_id, shift_start_time, shift_end_time, total_orders_processed, average_order_processing_time, last_shift_date)

Primary Key: cashier_id
Foreign Keys: cashier_id → EMPLOYEE(employee_id)

Hasil Tabel:

Field	Data Type	Constraint	Description
cashier_id	INT	PK, FK → EMPLOYEE	References employee_id
shift_start_time	TIME	NULL	Current shift start time
shift_end_time	TIME	NULL	Current shift end time
total_orders_processed	INT	NOT NULL, DEFAULT 0	Lifetime order count
average_order_processing_time	DECIMAL(5,2)	NULL	Avg time in minutes
last_shift_date	DATE	NULL	Most recent shift date

3. CHEF (Subtype)

Hasil Transformasi:

CHEF (chef_id, specialty, total_orders_prepared, average_prep_time,
orders_exceeding_sla)

Primary Key: chef_id
Foreign Keys: chef_id → EMPLOYEE(employee_id)

Hasil Tabel:

Field	Data Type	Constraint	Description
chef_id	INT	PK, FK → EMPLOYEE	References employee_id
specialty	ENUM	NOT NULL, DEFAULT 'GENERAL'	GRILL/FRYER/ASSEMBLY/DESSERT/GENERAL
total_orders_prepared	INT	NOT NULL, DEFAULT 0	Lifetime prepared orders
average_prep_time	DECIMAL(5,2)	NULL	Avg preparation time (minutes)
orders_exceeding_sla	INT	NOT NULL, DEFAULT 0	Orders over 30 minutes

4. WAITER (Subtype)

Hasil Transformasi:

WAITER (waiter_id, max_tables, current_table_count, total_deliveries,

average_delivery_time)

Primary Key: waiter_id
Foreign Keys: waiter_id → EMPLOYEE(employee_id)

Hasil Tabel:

Field	Data Type	Constraint	Description
waiter_id	INT	PK, FK → EMPLOYEE	References employee_id
max_tables	INT	NOT NULL, DEFAULT 6	Maximum concurrent tables
current_table_count	INT	NOT NULL, DEFAULT 0	Currently assigned tables
total_deliveries	INT	NOT NULL, DEFAULT 0	Lifetime delivery count
average_delivery_time	DECIMAL(5,2)	NULL	Avg delivery time (minutes)

5. MANAGER (Subtype)

Hasil Transformasi:

MANAGER (manager_id, authorization_level, total_refunds_approved,
total_refunds_denied)

Primary Key: manager_id
Foreign Keys: manager_id → EMPLOYEE(employee_id)

Hasil Tabel:

Field	Data Type	Constraint	Description
manager_id	INT	PK, FK → EMPLOYEE	References employee_id

authorization_level	ENUM	NOT NULL, DEFAULT 'SHIFT_MANAGER'	SHIFT_MANAGER/GENERAL_MANAGER/OWNER
total_refunds_approved	INT	NOT NULL, DEFAULT 0	Approved refund count
total_refunds_denied	INT	NOT NULL, DEFAULT 0	Denied refund count

6. CUSTOMER

Hasil Transformasi:

CUSTOMER (customer_id, phone_number, phone_number_hash, first_name, last_name,
total_stars, total_lifetime_orders, registration_date, last_order_date, is_vip, created_at, updated_at)

Primary Key: customer_id
Candidate Key: phone_number (UNIQUE but nullable for guest customers)
Foreign Keys: None

Hasil Tabel:

Field	Data Type	Constraint	Description
customer_id	INT	PK, AUTO_INCREMENT	Unique customer identifier
phone_number	VARCHAR(15)	UNIQUE, NULL	10-digit phone (NULL for guests)
phone_number_hash	CHAR(64)	NULL	Hashed phone for privacy
first_name	VARCHAR(50)	NULL	Customer first name
last_name	VARCHAR(50)	NULL	Customer last name

total_stars	INT	NOT NULL, DEFAULT 0	Current loyalty star balance
total_lifetime_orders	INT	NOT NULL, DEFAULT 0	Historical order count
registration_date	DATE	NOT NULL	Account creation date
last_order_date	DATE	NULL	Most recent order date
is_vip	BOOLEAN	NOT NULL, DEFAULT FALSE	TRUE when total_stars >= 100
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Record creation timestamp
updated_at	TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP	Last update timestamp

7. LOYALTY_STAR_TRANSACTION (Weak Entity)

Hasil Transformasi:

LOYALTY_STAR_TRANSACTION (star_transaction_id, customer_id, transaction_type,
star_amount, balance_before, balance_after, order_id, transaction_id, refund_id, discount_percentage, transaction_date, transaction_time, notes, created_by, created_at)

Primary Key: (star_transaction_id, customer_id) - Composite
Foreign Keys:

- customer_id → CUSTOMER(customer_id) - Identifying relationship
- order_id → ORDER(order_id)
- transaction_id → TRANSACTION(transaction_id)
- refund_id → REFUND(refund_id)
- created_by → EMPLOYEE(employee_id)

Hasil Tabel:

Field	Data Type	Constraint	Description
star_transaction_id	INT	PK (part 1), AUTO_INCREMENT	Transaction sequence number
customer_id	INT	PK (part 2), FK → CUSTOMER	Owner entity reference
transaction_type	ENUM	NOT NULL	EARNED/REDEEMED/REVERSED/ADJUSTED
star_amount	INT	NOT NULL	+/- stars (positive=earned, negative=redeemed)
balance_before	INT	NOT NULL	Star balance before transaction
balance_after	INT	NOT NULL	Star balance after transaction
order_id	INT	FK → ORDER, NULL	Linked order (if applicable)
transaction_id	INT	FK → TRANSACTION, NULL	Linked payment (if applicable)
refund_id	INT	FK → REFUND, NULL	Linked refund (if reversed)
discount_percentage	DECIMAL (5,2)	NULL	Discount % if redeemed (5/10/15/20)
transaction_date	DATE	NOT NULL	Transaction date
transaction_time	TIME	NOT NULL	Transaction time
notes	TEXT	NULL	Additional notes
created_by	INT	FK → EMPLOYEE, NULL	Employee who created (manual adjustments)

created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Record creation timestamp

8. MENU_ITEM

Hasil Transformasi:

MENU_ITEM (item_id, item_name, description, category, price, cost,
prep_time_minutes, is_available, is_active, image_url,
allergen_info, calories, created_at, updated_at, created_by)

Primary Key: item_id
Unique Constraint: (item_name, category) - unique within category
Foreign Keys: created_by → MANAGER(manager_id)

Hasil Tabel:

Field	Data Type	Constraint	Description
item_id	INT	PK, AUTO_INCREMENT	Unique menu item identifier
item_name	VARCHAR(100)	NOT NULL	Item display name
description	TEXT	NULL	Item description for menu
category	ENUM	NOT NULL	APPETIZER/MAIN/DESSERT/BEVERAGE
price	DECIMAL(10,2)	NOT NULL	Current selling price
cost	DECIMAL(10,2)	NULL	Cost of goods (for margin analysis)
prep_time_minutes	INT	NOT NULL	Estimated prep time (5-20 minutes)
is_available	BOOLEAN	NOT NULL, DEFAULT TRUE	Current availability flag

is_active	BOOLEAN	NOT NULL, DEFAULT TRUE	Soft delete flag
image_url	VARCHAR(255)	NULL	URL to menu item image
allergen_info	VARCHAR(255)	NULL	Allergen information
calories	INT	NULL	Calorie count
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Record creation timestamp
updated_at	TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP	Last update timestamp
created_by	INT	FK → MANAGER, NOT NULL	Manager who created item

9. RESTAURANT_TABLE

Hasil Transformasi:

RESTAURANT_TABLE (table_id, table_number, capacity, is_available,
current_order_id, assigned_waiter_id, occupied_since,
last_cleaned, location_zone, is_active)

Primary Key: table_id
Unique Key: table_number
Foreign Keys:

- assigned_waiter_id → WAITER(waiter_id)
- current_order_id → ORDER(order_id) - circular reference

Hasil Tabel:

Field	Data Type	Constraint	Description

table_id	INT	PK, AUTO_INCREMENT	Unique table identifier
table_number	INT	NOT NULL, UNIQUE	Display number (1-20)
capacity	INT	NOT NULL	Number of seats (2, 4, or 6)
is_available	BOOLEAN	NOT NULL, DEFAULT TRUE	Current availability status
current_order_id	INT	FK → ORDER, NULL	Currently occupying order
assigned_waiter_id	INT	FK → WAITER, NULL	Currently assigned waiter
occupied_since	TIMESTAMP	NULL	When customer sat down
last_cleaned	TIMESTAMP	NULL	Last cleaning timestamp
location_zone	ENUM	NOT NULL, DEFAULT 'MIDDLE'	FRONT/MIDDLE/BACK/PATIO
is_active	BOOLEAN	NOT NULL, DEFAULT TRUE	FALSE for maintenance

10. CUSTOMER_ORDER

Hasil Transformasi:

CUSTOMER_ORDER (order_id, order_number, customer_id, cashier_id, chef_id, waiter_id, table_id, order_date, order_time, order_type, status, status_updated_at, special_instructions, estimated_wait_time, actual_prep_time, subtotal, discount_amount, tax_amount, total_amount, stars_earned, stars_redeemed, is_flagged_for_review, created_at, queued_at, in_process_at, completed_at, delivered_at, related_order_id)

Primary Key: order_id
Unique Key: order_number
Foreign Keys:

- customer_id → CUSTOMER(customer_id)
- cashier_id → CASHIER(cashier_id)
- chef_id → CHEF(chef_id)
- waiter_id → WAITER(waiter_id)
- table_id → TABLE(table_id)
- related_order_id → ORDER(order_id) - self-reference

Hasil Tabel:

Field	Data Type	Constraint	Description
order_id	INT	PK, AUTO_INCREMENT	Unique order identifier
order_number	VARCHAR(20)	NOT NULL, UNIQUE	Business key (YYYYMMDD-####)
customer_id	INT	FK → CUSTOMER, NULL	NULL for guest orders
cashier_id	INT	FK → CASHIER, NOT NULL	Order entry cashier
chef_id	INT	FK → CHEF, NULL	Assigned when cooking starts
waiter_id	INT	FK → WAITER, NULL	For dine-in only
table_id	INT	FK → TABLE, NULL	NULL for takeout
order_date	DATE	NOT NULL	Order date

order_time	TIME	NOT NULL	Order time
order_type	ENUM	NOT NULL	DINE_IN/TAKEOUT
status	ENUM	NOT NULL, DEFAULT 'PENDING'	PENDING/QUEUED/IN_PROCESS/COMPLETED/DELIVERED/CANCELLED/REFUNDED
status_updated_at	TIMESTAMP	NULL	Last status change timestamp
special_instructions	TEXT	NULL	Order-level instructions (max 500 chars)
estimated_wait_time	INT	NOT NULL, DEFAULT 30	Estimated wait (minutes)
actual_prep_time	INT	NULL	Actual preparation time (minutes)
subtotal	DECIMAL(10, 2)	NOT NULL	Before discount and tax
discount_amount	DECIMAL(10, 2)	NOT NULL, DEFAULT 0.00	Loyalty discount applied
tax_amount	DECIMAL(10, 2)	NOT NULL	10% tax on (subtotal - discount)
total_amount	DECIMAL(10, 2)	NOT NULL	Final amount charged
stars_earned	INT	NOT NULL, DEFAULT 1	Stars credited (1 per order)
stars_redeemed	INT	NOT NULL, DEFAULT 0	Stars used for discount

is_flagged_for_review	BOOLEAN	NOT NULL, DEFAULT FALSE	Exceeds SLA or has issues
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Order creation timestamp
queued_at	TIMESTAMP	NULL	When sent to kitchen
in_process_at	TIMESTAMP	NULL	When chef started cooking
completed_at	TIMESTAMP	NULL	When chef finished
delivered_at	TIMESTAMP	NULL	When waiter delivered
related_order_id	INT	FK → ORDER, NULL	For split payment linking

11. ORDER_ITEM (Associative Entity)

Hasil Transformasi:

ORDER_ITEM (order_id, item_id, line_number, quantity, unit_price,
special_instructions, subtotal, created_at)

EER Concept: Associative entity for M:N relationship between ORDER and MENU_ITEM

Primary Key: (order_id, item_id, line_number) - Composite
Foreign Keys:

- order_id → ORDER(order_id)
- item_id → MENU_ITEM(item_id)

Hasil Tabel:

Field	Data Type	Constraint	Description

order_id	INT	PK (part 1), FK → ORDER	Order reference
item_id	INT	PK (part 2), FK → MENU_ITEM	Menu item reference
line_number	INT	PK (part 3)	Line sequence (1, 2, 3...)
quantity	INT	NOT NULL	Item quantity (1-99)
unit_price	DECIMAL(10, 2)	NOT NULL	Price snapshot at order time
special_instructions	VARCHAR(200)	NULL	Item-specific instructions
subtotal	DECIMAL(10, 2)	NOT NULL	quantity × unit_price
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Record creation timestamp

12. PAYMENT_TRANSACTION

Hasil Transformasi:

PAYMENT_TRANSACTION (transaction_id, order_id, transaction_date, transaction_time,
payment_method, subtotal, discount_amount, tax_amount, total_amount, amount_tendered, change_amount, card_last_four, card_authorization_code, card_type, transaction_status, processed_by, created_at)

Primary Key: transaction_id
Unique Key: order_id (one transaction per order)
Foreign Keys:

- order_id → ORDER(order_id)
- processed_by → CASHIER(cashier_id)

Hasil Tabel:

Field	Data Type	Constraint	Description
transaction_id	INT	PK, AUTO_INCREMENT	Unique transaction identifier
order_id	INT	FK → ORDER, NOT NULL, UNIQUE	One transaction per order
transaction_date	DATE	NOT NULL	Payment date
transaction_time	TIME	NOT NULL	Payment time
payment_method	ENUM	NOT NULL	CASH/CREDIT/DEBIT
subtotal	DECIMAL(10,2)	NOT NULL	Before discount and tax
discount_amount	DECIMAL(10,2)	NOT NULL, DEFAULT 0.00	Discount applied
tax_amount	DECIMAL(10,2)	NOT NULL	Tax amount (10%)
total_amount	DECIMAL(10,2)	NOT NULL	Final amount charged
amount_tendered	DECIMAL(10,2)	NULL	For CASH only
change_amount	DECIMAL(10,2)	NULL	For CASH only
card_last_four	CHAR(4)	NULL	Last 4 digits (PCI compliance)
card_authorization_code	VARCHAR(20)	NULL	Bank auth code for cards
card_type	ENUM	NULL	VISA/MASTERCARD/AMEX/DISCOVER

transaction_status	ENUM	NOT NULL, DEFAULT 'PENDING'	PENDING/COMPLETED/FAILED/REFUNDED
processed_by	INT	FK → CASHIER, NOT NULL	Cashier who processed
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Record creation timestamp

13. REFUND

Hasil Transformasi:

REFUND (refund_id, order_id, transaction_id, refund_date, refund_time,
refund_amount, refund_reason, reason_category, requested_by,
approved_by, approval_status, approval_date, refund_method,
stars_reversed, manager_notes, created_at, processed_at)

Primary Key: refund_id
Unique Key: order_id (one refund per order)
Foreign Keys:

- order_id → ORDER(order_id)
- transaction_id → TRANSACTION(transaction_id)
- requested_by → CASHIER(cashier_id)
- approved_by → MANAGER(manager_id)

Hasil Tabel:

Field	**Data Type**	**Constraint**	**Description**
refund_id	INT	PK, AUTO_INCREMENT	Unique refund identifier
order_id	INT	FK → ORDER, NOT NULL, UNIQUE	One refund per order

transaction_id	INT	FK → TRANSACTION, NOT NULL	Original payment reference
refund_date	DATE	NOT NULL	Refund request date
refund_time	TIME	NOT NULL	Refund request time
refund_amount	DECIMAL(10,2)	NOT NULL	Must equal original total
refund_reason	TEXT	NOT NULL	Detailed reason (min 10 chars)
reason_category	ENUM	NULL	EXCESSIVE_WAIT/WRONG_ORDER/QUALITY_ISSUE/CUSTOMER_REQUEST/OTHER
requested_by	INT	FK → CASHIER, NOT NULL	Cashier who initiated
approved_by	INT	FK → MANAGER, NOT NULL	Manager who approved
approval_status	ENUM	NOT NULL, DEFAULT 'PENDING'	PENDING/APPROVED/DENIED
approval_date	DATE	NULL	When decision made
refund_method	ENUM	NULL	ORIGINAL_PAYMENT/CASH/STORE_CREDIT
stars_reversed	INT		

14. AUDIT_LOG

Hasil Transformasi:

AUDIT_LOG (log_id, log_type, employee_id, action_description, table_affected,

record_id, ip_address, timestamp)

Primary Key: log_id
Foreign Keys: employee_id → EMPLOYEE(employee_id)

Hasil Tabel:

Field	Data Type	Constraint	Description
log_id	BIGINT	PK, AUTO_INCREMENT	Unique log entry identifier
log_type	ENUM	NOT NULL	LOGIN/LOGOUT/REFUND/MENU_CHANGE/OVERRIDE/ERROR
employee_id	INT	FK → EMPLOYEE, NULL	Employee who performed action
action_description	TEXT	NOT NULL	Detailed action description
table_affected	VARCHAR(50)	NULL	Database table name affected
record_id	INT	NULL	Primary key of affected record
ip_address	VARCHAR(45)	NULL	IPv4 or IPv6 address
timestamp	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	When action occurred

15. SYSTEM_CONFIG (SYSTEM TABLE)

Hasil Transformasi:

SYSTEM_CONFIG (config_key, config_value, data_type, description,
updated_by, updated_at)

Primary Key: config_key
Foreign Keys: updated_by → MANAGER(manager_id)

Hasil Tabel:

Field	Data Type	Constraint	Description
config_key	VARCHAR(50)	PK	Unique configuration parameter name
config_value	TEXT	NOT NULL	Parameter value (stored as text)
data_type	ENUM	NOT NULL	STRING/INTEGER/DECIMAL/BOOLEAN
description	TEXT	NULL	Configuration parameter description
updated_by	INT	FK → MANAGER, NULL	Manager who last updated
updated_at	TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP	Last update timestamp

Referential Integrity:

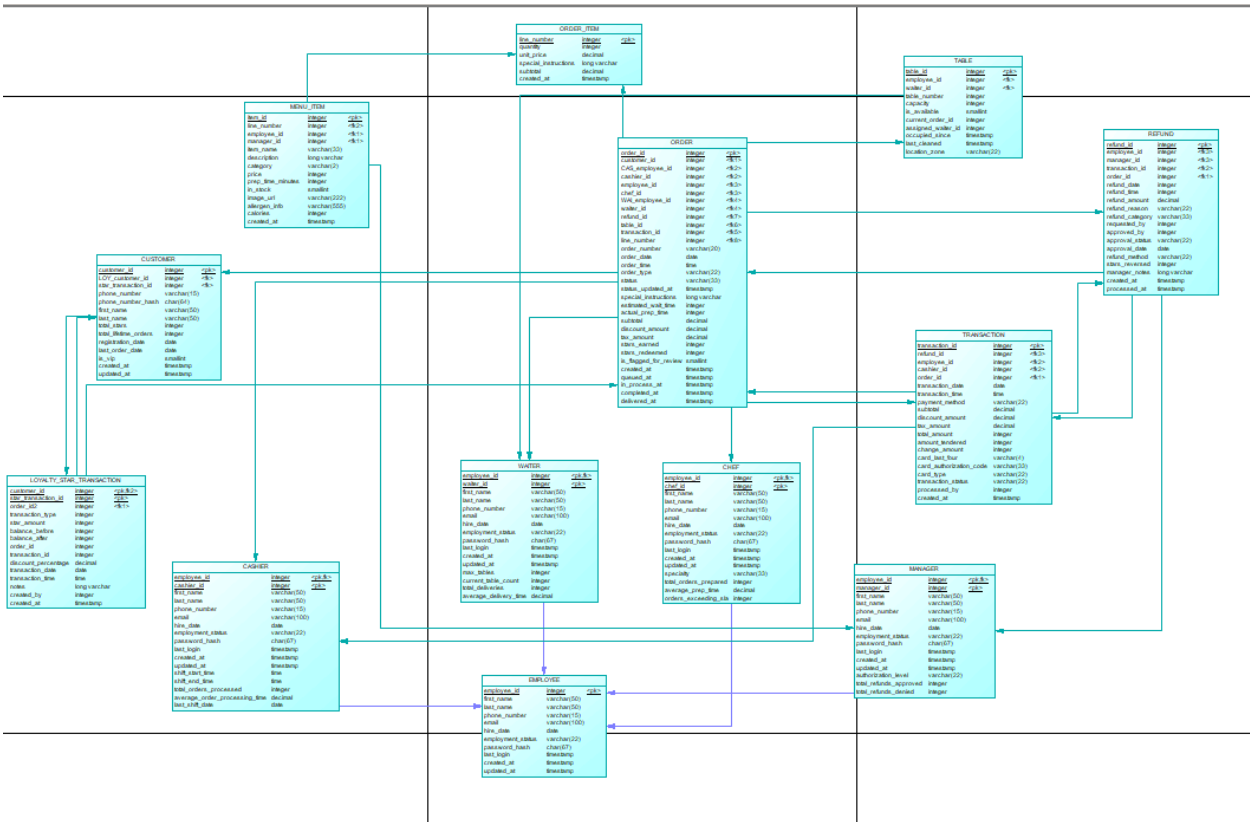
- updated_by: ON DELETE SET NULL, ON UPDATE CASCADE

Sample Configuration Values:

config_key	config_value	data_type	description
tax_rate	0.10	DECIMAL	Sales tax rate (10%)
order_sla_minutes	30	INTEGER	Order completion SLA
max_items_per_order	50	INTEGER	Maximum items per order
min_order_value	5.00	DECIMAL	Minimum order subtotal
max_waiter_tables	6	INTEGER	Max tables per waiter

star_10_discount	5.00	DECIMAL	10-star discount %
star_25_discount	10.00	DECIMAL	25-star discount %
star_50_discount	15.00	DECIMAL	50-star discount %
star_100_discount	20.00	DECIMAL	100-star discount %

HASIL RELASI TABEL :

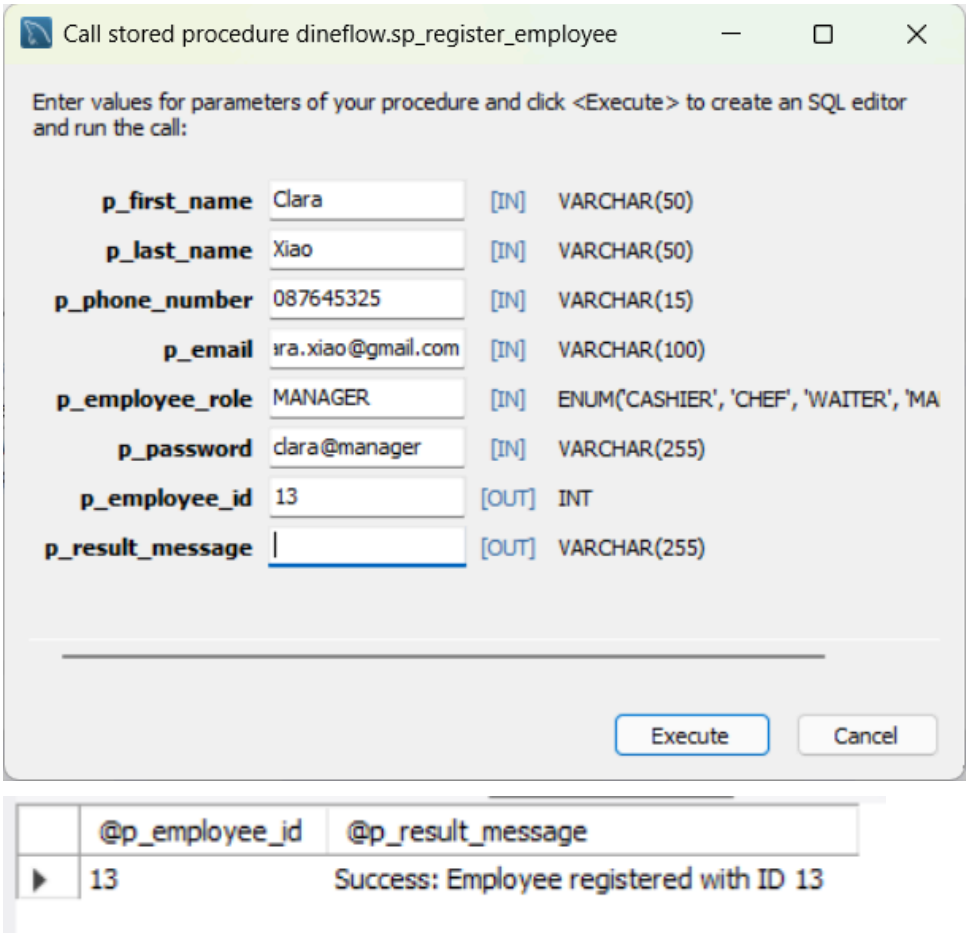


note: Tabel audit_log dan tabel system_config lock tidak ditampilkan dalam diagram relasi tabel (PDM) karena merupakan tabel sistem yang digunakan untuk keperluan internal dan implementasi sistem, bukan bagian dari data bisnis utama.

DESAIN STORE PROCEDURE

1. Procedure: sp_register_employee

Nama	sp_register_employee
Tabel dan Field yang Terlibat	**EMPLOYEE table:** • employee_id (PK, AUTO_INCREMENT) • first_name, last_name • phone_number (UNIQUE) • email (UNIQUE) • hire_date, employment_status • employee_role, password_hash **Role-specific tables:** • CASHIER (cashier_id FK) • CHEF (chef_id FK, specialty) • WAITER (waiter_id FK, max_tables) • MANAGER (manager_id FK, authorization_level)
Mode dan Deskripsi Fungsi SP	**Mode:** INSERT **Deskripsi:** Mendaftarkan karyawan baru ke dalam sistem dengan validasi: 1. Memvalidasi uniqueness phone_number dan email 2. Memvalidasi format phone (10-15 digit) 3. Meng-hash password menggunakan SHA2-256 4. Insert ke tabel EMPLOYEE (supertype) 5. Insert ke tabel role-specific sesuai employee_role (subtype) 6. Implementasi ISA relationship (Generalization/Specialization) **Business Rules:** • Phone harus unique dan 10-15 digit • Email harus unique • Password di-hash untuk keamanan • Setiap employee harus punya 1 role (Total participation) • Tidak boleh duplicate phone/email
Output	**OUT p_employee_id (INT):** • NULL jika gagal • Employee ID baru jika sukses

	OUT p_result_message (VARCHAR(255)): • 'Success: Employee registered with ID X' • 'Error: Phone number already registered.' • 'Error: Email already registered.' • 'Error: Phone must be 10-15 digits.' • 'Error: Failed to register employee.'
	

Query:

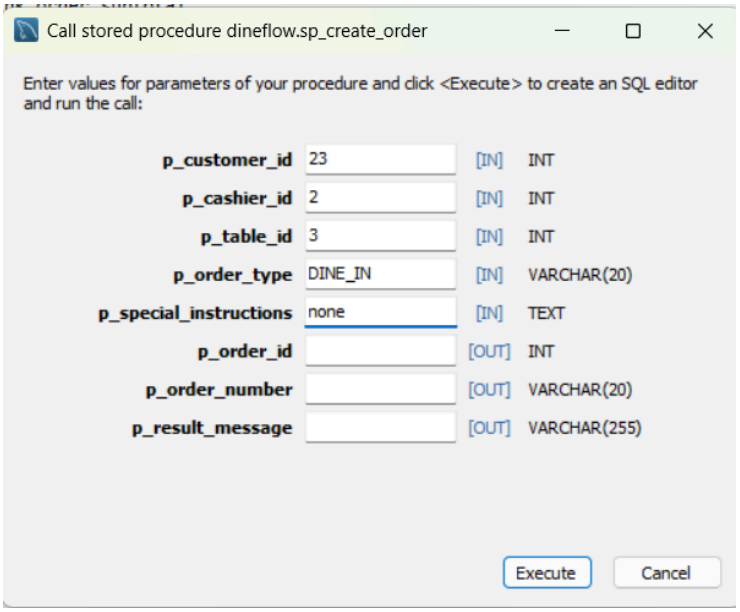
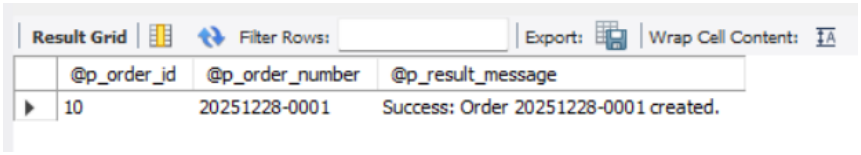
```
CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_register_employee`(
    IN p_first_name VARCHAR(50),
    IN p_last_name VARCHAR(50),
    IN p_phone_number VARCHAR(15),
    IN p_email VARCHAR(100),
    IN p_employee_role ENUM('CASHIER', 'CHEF', 'WAITER', 'MANAGER'),
    IN p_password VARCHAR(255),
    OUT p_employee_id INT,
    OUT p_result_message VARCHAR(255)
)
BEGIN
    DECLARE v_existing_phone INT DEFAULT 0;
    DECLARE v_existing_email INT DEFAULT 0;
    DECLARE v_password_hash CHAR(64);

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SET p_employee_id = NULL;
        SET p_result_message = 'Error: Employee ID cannot be null.';

    START TRANSACTION;

    -- Check if phone already exists
    SELECT COUNT(*) INTO v_existing_phone
    FROM employee
    WHERE phone_number = p_phone_number;

    -- Check if email already exists
    SELECT COUNT(*) INTO v_existing_email
    FROM employee
    WHERE email = p_email;

    IF v_existing_phone > 0 THEN
        SET p_employee_id = NULL;
        SET p_result_message = 'Error: Phone number already registered.';
        ROLLBACK;

    ELSEIF v_existing_email > 0 THEN
        SET p_employee_id = NULL;
        SET p_result_message = 'Error: Email already registered.';
        ROLLBACK;
```

```
    ELSEIF p_phone_number NOT REGEXP '^[0-9]{10,15}$' THEN
        SET p_employee_id = NULL;
        SET p_result_message = 'Error: Phone must be 10-15 digits.';
        ROLLBACK;

    ELSE
        -- Hash the password
        SET v_password_hash = SHA2(p_password, 256);

        -- Insert into EMPLOYEE table
        INSERT INTO employee (
            first_name, last_name, phone_number, email,
            hire_date, employment_status, employee_role,
            password_hash
        ) VALUES (
            p_first_name, p_last_name, p_phone_number, p_email,
            CURDATE(), 'ACTIVE', p_employee_role,
            v_password_hash
        );

        SET p_employee_id = LAST_INSERT_ID();


       -- Insert into role-specific table based on employee_role
       CASE p_employee_role
           WHEN 'CASHIER' THEN
               INSERT INTO cashier (cashier_id) VALUES (p_employee_id);

           WHEN 'CHEF' THEN
               INSERT INTO chef (chef_id, specialty) VALUES (p_employee_id, 'GENERAL');

           WHEN 'WAITER' THEN
               INSERT INTO waiter (waiter_id, max_tables) VALUES (p_employee_id, 6);

           WHEN 'MANAGER' THEN
               INSERT INTO manager (manager_id, authorization_level)
               VALUES (p_employee_id, 'SHIFT_MANAGER');
       END CASE;

       SET p_result_message = CONCAT('Success: Employee registered with ID ', p_employee_id)
       COMMIT;
    END IF;
END
```

Procedure 2: sp_create_order

Attribute	Details
Nama	sp_create_order

Tabel dan Field yang Terlibat	**CUSTOMER_ORDER table:** • order_id (PK, AUTO_INCREMENT) • order_number (UNIQUE, generated) • customer_id (FK → CUSTOMER, nullable) • cashier_id (FK → CASHIER) • table_id (FK → RESTAURANT_TABLE, nullable) • order_date, order_time • order_type (ENUM: DINE_IN/TAKEOUT) • status (default: PENDING) • special_instructions • subtotal, tax_amount, total_amount **RESTAURANT_TABLE table:** • table_id, is_available • current_order_id, occupied_since **CASHIER + EMPLOYEE tables:** • Untuk validasi cashier aktif
Mode dan Deskripsi Fungsi SP	**Mode:** INSERT + UPDATE **Deskripsi:** Membuat order baru dengan validasi lengkap: 1. Validasi cashier exists dan ACTIVE 2. Untuk DINE_IN: validasi table_id required dan table available 3. Generate order_number format YYYYMMDD-#### (e.g., 20241227-0001) 4. Insert order baru dengan status PENDING 5. Jika DINE_IN: update table menjadi occupied **Business Rules:** • DINE_IN wajib punya table_id • TAKEOUT tidak perlu table_id • Order number unique per hari (sequence reset harian) • Initial amounts = 0 (items ditambah kemudian) • Table harus available untuk dine-in
Output	**OUT p_order_id (INT):** • NULL jika gagal • Order ID baru jika sukses **OUT p_order_number (VARCHAR(20)):** • Format: YYYYMMDD-#### (e.g., 20241227-0001)

	OUT p_result_message (VARCHAR(255)): • 'Success: Order 20241227-0001 created.' • 'Error: Cashier not found or inactive.' • 'Error: Table required for dine-in orders.' • 'Error: Table not found.' • 'Error: Table is currently occupied.' • 'Error: Failed to create order.'
	 

Query:

```
use dineflow;
DELIMITER $$

CREATE PROCEDURE sp_create_order (
    IN  p_customer_id INT,
    IN  p_cashier_id INT,
    IN  p_table_id INT,
    IN  p_order_type VARCHAR(20),
    IN  p_special_instructions TEXT,
    OUT p_order_id INT,
    OUT p_order_number VARCHAR(20),
    OUT p_result_message VARCHAR(255)
)
proc: BEGIN
    DECLARE v_cashier_exists INT;
    DECLARE v_table_available BOOLEAN;
    DECLARE v_order_count INT;
    DECLARE v_order_num VARCHAR(20);
```

```
    -- Diagnostic variables
    DECLARE v_errno INT;
    DECLARE v_msg TEXT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        GET DIAGNOSTICS CONDITION 1
            v_errno = MYSQL_ERRNO,
            v_msg   = MESSAGE_TEXT;

        ROLLBACK;

        SET p_order_id = NULL;
        SET p_order_number = NULL;
        SET p_result_message = CONCAT(
            'DB ERROR ', v_errno, ': ', v_msg
        );
    END;
```

```
    SET p_order_id = NULL;
    SET p_order_number = NULL;
    SET p_result_message = NULL;

    IF p_order_type NOT IN ('DINE_IN', 'TAKEOUT') THEN
        SET p_result_message = 'Error: Invalid order type.';
        LEAVE proc;
    END IF;

    SELECT COUNT(*)
    INTO v_cashier_exists
    FROM cashier c
    JOIN employee e ON e.employee_id = c.cashier_id
    WHERE c.cashier_id = p_cashier_id
      AND e.employment_status = 'ACTIVE';

    IF v_cashier_exists = 0 THEN
        SET p_result_message = 'Error: Cashier not found or inactive.';
        LEAVE proc;
    END IF;

    IF p_order_type = 'DINE_IN' THEN
```

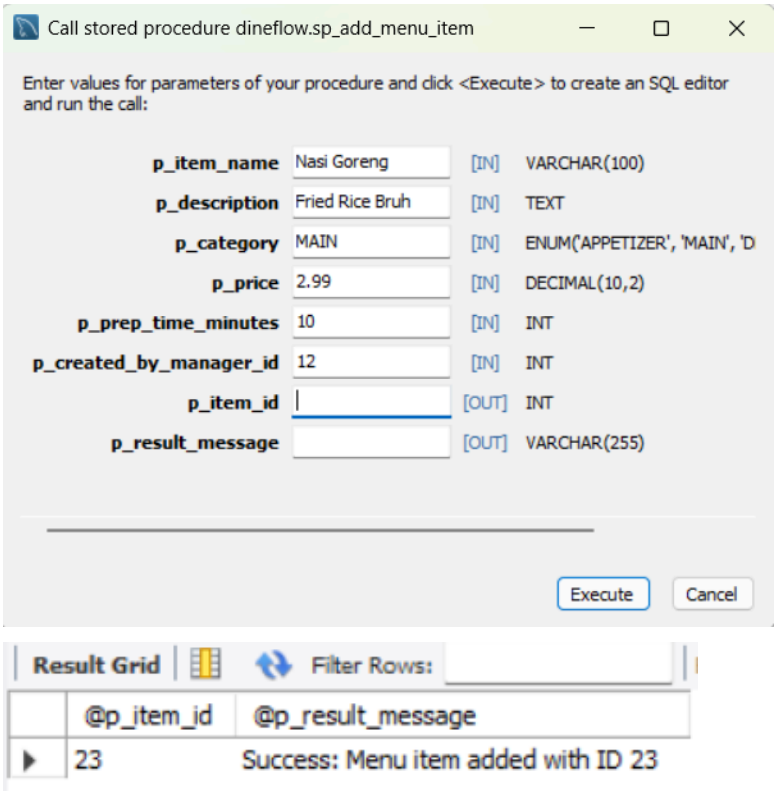
```
  IF p_order_type = 'DINE_IN' THEN

      IF p_table_id IS NULL THEN
          SET p_result_message = 'Error: Table required for dine-in orders.';
          LEAVE proc;
      END IF;

      SELECT is_available
      INTO v_table_available
      FROM restaurant_table
      WHERE table_id = p_table_id
        AND is_active = TRUE;

      IF v_table_available IS NULL THEN
          SET p_result_message = 'Error: Table not found.';
          LEAVE proc;
      END IF;

      IF v_table_available = FALSE THEN
          SET p_result_message = 'Error: Table is currently occupied.';
          LEAVE proc;
      END IF;
```

```
START TRANSACTION;

SELECT COUNT(*)
INTO v_order_count
FROM customer_order
WHERE order_date = CURDATE();

SET v_order_num = CONCAT(
    DATE_FORMAT(CURDATE(), '%Y%m%d'),
    '-',
    LPAD(v_order_count + 1, 4, '0')
);

INSERT INTO customer_order (
    order_number,
    customer_id,
    cashier_id,
    table_id,
    order_date,
    order_time,
    order_type,
    status,
    special_instructions,


     special_instructions,
     subtotal,
     tax_amount,
     total_amount
)
VALUES (
     v_order_num,
     p_customer_id,
     p_cashier_id,
     p_table_id,
     CURDATE(),
     CURTIME(),
     p_order_type,
     'PENDING',
     p_special_instructions,
     0.00,
     0.00,
     0.00
);

SET p_order_id = LAST_INSERT_ID();
SET p_order_number = v_order_num;
```

```
    IF p_order_type = 'DINE_IN' THEN
        UPDATE restaurant_table
        SET is_available = FALSE,
            current_order_id = p_order_id,
            occupied_since = NOW()
        WHERE table_id = p_table_id;
    END IF;

    COMMIT;

    SET p_result_message = CONCAT(
        'Success: Order ',
        v_order_num,
        ' created.'
    );

END$$

DELIMITER ;
```

Procedure 3: sp_add_menu_item

Attribute	Details
Nama	sp_add_menu_item
Tabel dan Field yang Terlibat	**MENU_ITEM table:** • item_id (PK, AUTO_INCREMENT) • item_name, description • category (ENUM: APPETIZER/MAIN/DESSERT/BEVERAGE) • price, prep_time_minutes • is_available (default: TRUE) • is_active (default: TRUE) • created_by (FK → MANAGER) **MANAGER + EMPLOYEE tables:** • Untuk validasi manager aktif **UNIQUE Constraint:** • (item_name, category) - mencegah duplicate dalam kategori yang sama
Mode dan Deskripsi Fungsi SP	**Mode:** INSERT **Deskripsi:** Menambahkan menu item baru oleh manager dengan validasi: 1. Validasi manager exists dan ACTIVE 2. Validasi price > 0

	3. Validasi prep_time antara 5-20 menit (business rule) 4. Check duplicate: item_name + category harus unique 5. Insert menu item baru sebagai available dan active **Business Rules:** • Hanya manager yang bisa add menu item • Price harus > 0 • Prep time: 5-20 menit (standar fast food) • Tidak boleh duplicate name dalam kategori sama • Item dimulai sebagai available=TRUE, active=TRUE
Output	**OUT p_item_id (INT):** • NULL jika gagal • Menu item ID baru jika sukses **OUT p_result_message (VARCHAR(255)):** • 'Success: Menu item added with ID X' • 'Error: Manager not found or inactive.' • 'Error: Price must be greater than 0.' • 'Error: Prep time must be between 5 and 20 minutes.' • 'Error: Item already exists in this category.' • 'Error: Failed to add menu item.'
	

Query:

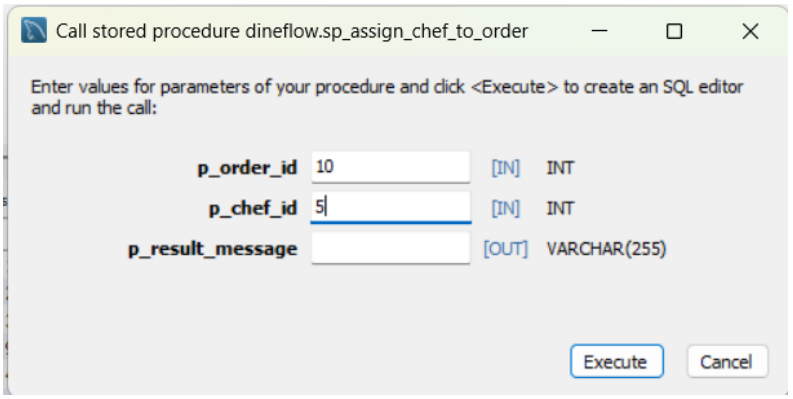
```
use dineflow;
DELIMITER $$

CREATE PROCEDURE sp_create_order (
    IN  p_customer_id INT,
    IN  p_cashier_id INT,
    IN  p_table_id INT,
    IN  p_order_type VARCHAR(20),
    IN  p_special_instructions TEXT,
    OUT p_order_id INT,
    OUT p_order_number VARCHAR(20),
    OUT p_result_message VARCHAR(255)
)
proc: BEGIN
    DECLARE v_cashier_exists INT;
    DECLARE v_table_available BOOLEAN;
    DECLARE v_order_count INT;
    DECLARE v_order_num VARCHAR(20);

    -- Diagnostic variables
    DECLARE v_errno INT;
    DECLARE v_msg TEXT;

    -- Diagnostic variables
    DECLARE v_errno INT;
    DECLARE v_msg TEXT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        GET DIAGNOSTICS CONDITION 1
            v_errno = MYSQL_ERRNO,
            v_msg   = MESSAGE_TEXT;

        ROLLBACK;

        SET p_order_id = NULL;
        SET p_order_number = NULL;
        SET p_result_message = CONCAT(
            'DB ERROR ', v_errno, ': ', v_msg
        );
    END;

    -- Default outputs
    SET p_order_id = NULL;
    SET p_order_number = NULL;
    SET p_result_message = NULL;
```

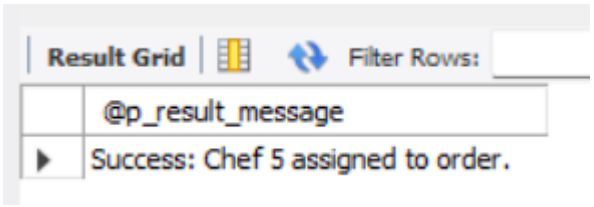
```
IF p_order_type NOT IN ('DINE_IN', 'TAKEOUT') THEN
    SET p_result_message = 'Error: Invalid order type.';
    LEAVE proc;
END IF;

SELECT COUNT(*)
INTO v_cashier_exists
FROM cashier c
JOIN employee e ON e.employee_id = c.cashier_id
WHERE c.cashier_id = p_cashier_id
  AND e.employment_status = 'ACTIVE';

IF v_cashier_exists = 0 THEN
    SET p_result_message = 'Error: Cashier not found or inactive.';
    LEAVE proc;
END IF;

IF p_order_type = 'DINE_IN' THEN

    IF p_table_id IS NULL THEN
        SET p_result_message = 'Error: Table required for dine-in orders.';
        LEAVE proc;

     SELECT is_available
     INTO v_table_available
     FROM restaurant_table
     WHERE table_id = p_table_id
       AND is_active = TRUE;

     IF v_table_available IS NULL THEN
         SET p_result_message = 'Error: Table not found.';
         LEAVE proc;
     END IF;

     IF v_table_available = FALSE THEN
         SET p_result_message = 'Error: Table is currently occupied.';
         LEAVE proc;
     END IF;

 END IF;
 START TRANSACTION;

 SELECT COUNT(*)
 INTO v_order_count
 FROM customer_order
 WHERE order_date = CURDATE();
```

```
    SET v_order_num = CONCAT(
        DATE_FORMAT(CURDATE(), '%Y%m%d'),
        '-',
        LPAD(v_order_count + 1, 4, '0')
    );

    INSERT INTO customer_order (
        order_number,
        customer_id,
        cashier_id,
        table_id,
        order_date,
        order_time,
        order_type,
        status,
        special_instructions,
        subtotal,
        tax_amount,
        total_amount
    )
    VALUES (
        v_order_num,
        p_customer_id,
        p_cashier_id,
        p_table_id,
        CURDATE(),
        CURTIME(),
        p_order_type,
        'PENDING',
        p_special_instructions,
        0.00,
        0.00,
        0.00
    );

    SET p_order_id = LAST_INSERT_ID();
    SET p_order_number = v_order_num;

    IF p_order_type = 'DINE_IN' THEN
        UPDATE restaurant_table
        SET is_available = FALSE,
            current_order_id = p_order_id,
            occupied_since = NOW()
        WHERE table_id = p_table_id;
    END IF;


    COMMIT;

    SET p_result_message = CONCAT(
        'Success: Order ',
        v_order_num,
        ' created.'
    );

END$$

DELIMITER ;
```

Procedure 4: sp_assign_chef_to_order

Attribute	Details
Nama	sp_assign_chef_to_order
Tabel dan Field yang Terlibat	**CUSTOMER_ORDER table:** • order_id (PK) • chef_id (FK → CHEF) • status (PENDING → IN_PROCESS) • status_updated_at • in_process_at (timestamp cooking dimulai) **CHEF + EMPLOYEE tables:** • Untuk validasi chef aktif
Mode dan Deskripsi Fungsi SP	**Mode:** UPDATE **Deskripsi:** Assign chef ke order dan mulai proses memasak: 1. Validasi order exists 2. Validasi order status = PENDING atau QUEUED 3. Validasi chef exists dan ACTIVE 4. Update order: assign chef_id 5. Update status: PENDING/QUEUED → IN_PROCESS 6. Record timestamp in_process_at (untuk tracking waktu) **Business Rules:** • Hanya order PENDING/QUEUED yang bisa diassign • Order IN_PROCESS/COMPLETED tidak bisa reassign • Chef harus ACTIVE employee • Timestamp penting untuk SLA tracking (30 menit max) • State transition: PENDING → QUEUED → IN_PROCESS
Output	**OUT p_result_message (VARCHAR(255)):** • 'Success: Chef X assigned to order.' • 'Error: Order not found.' • 'Error: Order already in progress or completed.' • 'Error: Chef not found or inactive.' • 'Error: Failed to assign chef.'





Query:

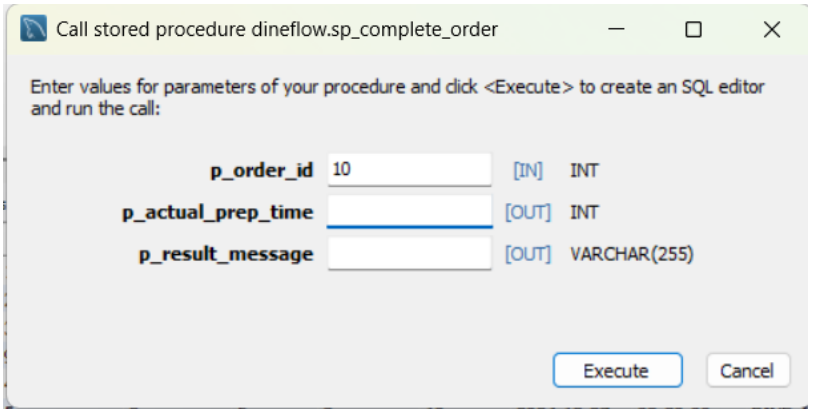
```
use dineflow;
DELIMITER $$
CREATE PROCEDURE sp_assign_chef_to_order(
    IN p_order_id INT,
    IN p_chef_id INT,
    OUT p_result_message VARCHAR(255)
)
BEGIN
    DECLARE v_order_status VARCHAR(20);
    DECLARE v_chef_exists INT DEFAULT 0;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SET p_result_message = 'Error: Failed to assign chef.';
    END;

    START TRANSACTION;
    -- Check order exists and status
    SELECT status INTO v_order_status
    FROM customer_order
    WHERE order_id = p_order_id;
```

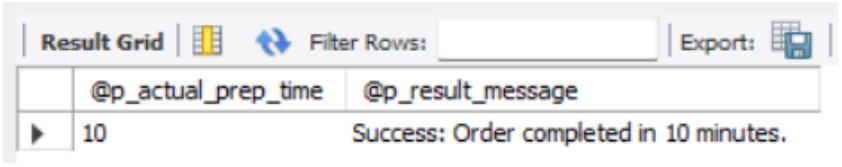
```
    IF v_order_status IS NULL THEN
        SET p_result_message = 'Error: Order not found.';
        ROLLBACK;

    ELSEIF v_order_status NOT IN ('PENDING', 'QUEUED') THEN
        SET p_result_message = 'Error: Order already in progress or completed.';
        ROLLBACK;

    ELSE
        -- Verify chef exists and is active
        SELECT COUNT(*) INTO v_chef_exists
        FROM chef c
        JOIN employee e ON c.chef_id = e.employee_id
        WHERE c.chef_id = p_chef_id
          AND e.employment_status = 'ACTIVE';

        IF v_chef_exists = 0 THEN
            SET p_result_message = 'Error: Chef not found or inactive.';
            ROLLBACK;
        ELSE
            -- Assign chef and update status
            UPDATE customer_order
            SET chef_id = p_chef_id,

                status = 'IN_PROCESS',
                status_updated_at = NOW(),
                in_process_at = NOW()
             WHERE order_id = p_order_id;

             SET p_result_message = CONCAT('Success: Chef ', p_chef_id, ' assigned to order.');
             COMMIT;
         END IF;
    END IF;
END$$

DELIMITER ;
```

Procedure 5: sp_complete_order

Attribute	Details
Nama	sp_complete_order
Tabel dan Field yang Terlibat	**CUSTOMER_ORDER table:** • order_id (PK) • status (IN_PROCESS → COMPLETED) • in_process_at (untuk kalkulasi) • completed_at (timestamp selesai) • actual_prep_time (hasil kalkulasi) • estimated_wait_time (untuk comparison) • is_flagged_for_review (auto-flag jika lambat) • status_updated_at **CHEF table:** • chef_id (PK)

	• total_orders_prepared (increment) • average_prep_time (recalculate running average) • orders_exceeding_sla (increment jika > 30 menit)
Mode dan Deskripsi Fungsi SP	**Mode:** UPDATE (multiple tables) **Deskripsi:** Menyelesaikan order dan update statistik chef: 1. Validasi order status = IN_PROCESS 2. Hitung actual prep time: TIMESTAMPDIFF(in_process_at, NOW()) 3. Update order: status → COMPLETED, save prep time 4. Auto-flag jika actual > estimated (quality control) 5. Update chef statistics: • total_orders_prepared + 1 • Recalculate average_prep_time (running average) • Increment orders_exceeding_sla jika > 30 menit **Business Rules:** • Hanya order IN_PROCESS yang bisa completed • SLA = 30 menit (orders > 30 min di-flag) • Auto-flag untuk review jika lambat dari estimasi • Running average formula: (old_avg × count + new) / (count + 1) • Performance tracking untuk chef evaluation
Output	**OUT p_actual_prep_time (INT):** • NULL jika gagal • Jumlah menit actual prep time jika sukses **OUT p_result_message (VARCHAR(255)):** • 'Success: Order completed in X minutes.' • 'Error: Order not found.' • 'Error: Order is not in process.' • 'Error: Order has no start time.' • 'Error: Failed to complete order.'






```
use dineflow;
DELIMITER $$

CREATE PROCEDURE sp_complete_order(
    IN p_order_id INT,
    OUT p_actual_prep_time INT,
    OUT p_result_message VARCHAR(255)
)
BEGIN
    DECLARE v_order_status VARCHAR(20);
    DECLARE v_in_process_time TIMESTAMP;
    DECLARE v_prep_minutes INT;
    DECLARE v_chef_id INT;
    DECLARE v_estimated_wait INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SET p_result_message = 'Error: Failed to complete order.';
    END;

    START TRANSACTION;
```

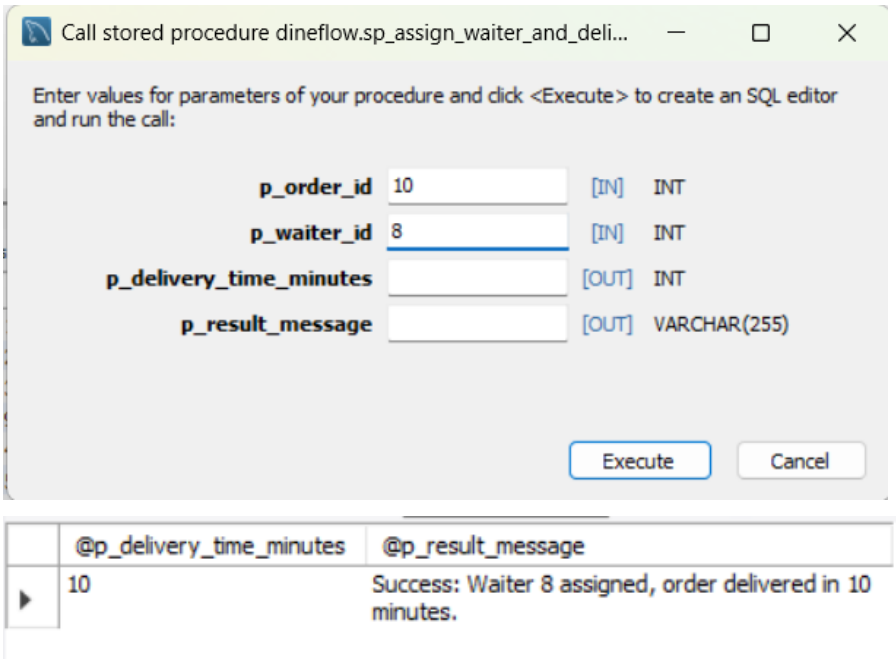
```
    SELECT status, in_process_at, chef_id, estimated_wait_time
    INTO v_order_status, v_in_process_time, v_chef_id, v_estimated_wait
    FROM customer_order
    WHERE order_id = p_order_id;

    IF v_order_status IS NULL THEN
        SET p_result_message = 'Error: Order not found.';
        ROLLBACK;

    ELSEIF v_order_status != 'IN_PROCESS' THEN
        SET p_result_message = 'Error: Order is not in process.';
        ROLLBACK;

    ELSEIF v_in_process_time IS NULL THEN
        SET p_result_message = 'Error: Order has no start time.';
        ROLLBACK;

    ELSE
        -- Calculate actual prep time in minutes
        SET v_prep_minutes = TIMESTAMPDIFF(MINUTE, v_in_process_time, NOW());

        -- Update order to completed
        UPDATE customer_order
```

```
        UPDATE customer_order
        SET status = 'COMPLETED',
            status_updated_at = NOW(),
            completed_at = NOW(),
            actual_prep_time = v_prep_minutes,
            is_flagged_for_review = (v_prep_minutes > v_estimated_wait)
        WHERE order_id = p_order_id;

        -- Update chef statistics
        UPDATE chef
        SET total_orders_prepared = total_orders_prepared + 1,
            average_prep_time = (
                (COALESCE(average_prep_time, 0) * total_orders_prepared + v_prep_minutes)
                / (total_orders_prepared + 1)
            ),
            orders_exceeding_sla = orders_exceeding_sla +
                CASE WHEN v_prep_minutes > 30 THEN 1 ELSE 0 END
        WHERE chef_id = v_chef_id;

        SET p_actual_prep_time = v_prep_minutes;
        SET p_result_message = CONCAT('Success: Order completed in ', v_prep_minutes, ' minutes.');
        COMMIT;
    END IF;
```

Procedure 6: sp_assign_waiter_and_deliver_order

Attribute	Details
Nama	sp_assign_waiter_and_deliver_order
Tabel dan Field yang Terlibat	**CUSTOMER_ORDER table:** • order_id (PK) • waiter_id (FK → WAITER) • status (COMPLETED → DELIVERED) • delivered_at (timestamp delivery) • status_updated_at • order_type (validasi: hanya DINE_IN) • table_id (untuk link ke table)

	WAITER table: • waiter_id (PK) • current_table_count (increment/decrement) • max_tables (untuk validasi kapasitas) • total_deliveries (increment) • average_delivery_time (recalculate) **RESTAURANT_TABLE table:** • table_id (PK) • assigned_waiter_id (link waiter ke table) **EMPLOYEE table:** • employment_status (validasi ACTIVE)
Mode dan Deskripsi Fungsi SP	**Mode:** UPDATE (multiple tables) **Deskripsi:** Assign waiter untuk deliver order ke customer dan update statistics: 1. Validasi order exists dan status = COMPLETED 2. Validasi order_type = DINE_IN (takeout tidak butuh waiter) 3. Validasi waiter exists, ACTIVE, dan belum full capacity 4. Check waiter capacity: current_table_count < max_tables 5. Assign waiter ke order dan table 6. Update order status: COMPLETED → DELIVERED 7. Update waiter stats: • total_deliveries + 1 • current_table_count + 1 • Recalculate average_delivery_time 8. Link waiter ke table (assigned_waiter_id) **Business Rules:** • Hanya DINE_IN orders butuh waiter delivery • Waiter max 6 tables concurrent (default, customizable) • Delivery time = completed_at sampai delivered_at • Waiter harus ACTIVE employee • State: COMPLETED → DELIVERED • Track delivery performance untuk evaluation
Output	**OUT p_delivery_time_minutes (INT):** • NULL jika gagal • Delivery time dalam menit (completed → delivered) **OUT p_result_message (VARCHAR(255)):**

	• 'Success: Waiter X assigned, order delivered in Y minutes.' • 'Error: Order not found.' • 'Error: Order not ready for delivery (must be COMPLETED).' • 'Error: Takeout orders do not require waiter delivery.' • 'Error: Waiter not found or inactive.' • 'Error: Waiter at full capacity (max tables reached).' • 'Error: Failed to assign waiter.'
	 @p_delivery_time_minutes: 10 @p_result_message: Success: Waiter 8 assigned, order delivered in 10 minutes.

Query:

```
DELIMITER $$

CREATE PROCEDURE sp_assign_waiter_and_deliver_order(
    IN p_order_id INT,
    IN p_waiter_id INT,
    OUT p_delivery_time_minutes INT,
    OUT p_result_message VARCHAR(255)
)
BEGIN
    DECLARE v_order_status VARCHAR(20);
    DECLARE v_order_type VARCHAR(20);
    DECLARE v_table_id INT;
    DECLARE v_completed_at TIMESTAMP;
    DECLARE v_waiter_exists INT DEFAULT 0;
    DECLARE v_current_tables INT;
    DECLARE v_max_tables INT;
    DECLARE v_delivery_minutes INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SET p_delivery_time_minutes = NULL;
        SET p_result_message = 'Error: Failed to assign waiter.';
    START TRANSACTION;
    -- Get order details
    SELECT status, order_type, table_id, completed_at
    INTO v_order_status, v_order_type, v_table_id, v_completed_at
    FROM customer_order
    WHERE order_id = p_order_id;

    IF v_order_status IS NULL THEN
        SET p_delivery_time_minutes = NULL;
        SET p_result_message = 'Error: Order not found.';
        ROLLBACK;

    ELSEIF v_order_status != 'COMPLETED' THEN
        SET p_delivery_time_minutes = NULL;
        SET p_result_message = 'Error: Order not ready for delivery (must be COMPLETED).';
        ROLLBACK;

    ELSEIF v_order_type = 'TAKEOUT' THEN
        SET p_delivery_time_minutes = NULL;
        SET p_result_message = 'Error: Takeout orders do not require waiter delivery.';
        ROLLBACK;
```

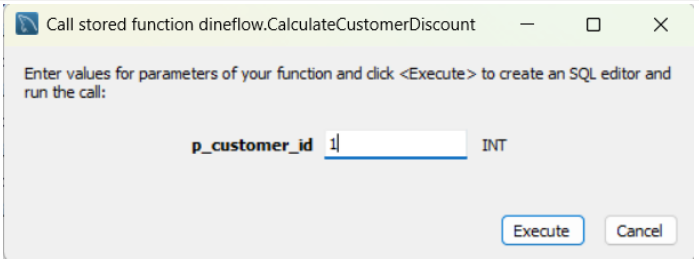
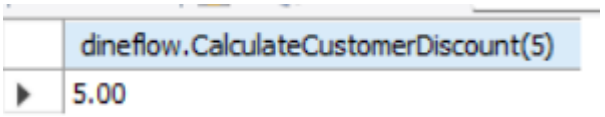
```
    ELSEIF v_completed_at IS NULL THEN
        SET p_delivery_time_minutes = NULL;
        SET p_result_message = 'Error: Order has no completion time.';
        ROLLBACK;

    ELSE
        -- Verify waiter exists, is active, and check capacity
        SELECT COUNT(*), w.current_table_count, w.max_tables
        INTO v_waiter_exists, v_current_tables, v_max_tables
        FROM waiter w
        JOIN employee e ON w.waiter_id = e.employee_id
        WHERE w.waiter_id = p_waiter_id
          AND e.employment_status = 'ACTIVE'
        GROUP BY w.current_table_count, w.max_tables;

        IF v_waiter_exists = 0 THEN
            SET p_delivery_time_minutes = NULL;
            SET p_result_message = 'Error: Waiter not found or inactive.';
            ROLLBACK;

        ELSEIF v_current_tables >= v_max_tables THEN
            SET p_delivery_time_minutes = NULL;
            SET p_result_message = 'Error: Waiter at full capacity (max tables reached).';

        ELSE
            -- Calculate delivery time
            SET v_delivery_minutes = TIMESTAMPDIFF(MINUTE, v_completed_at, NOW());

            -- Update order: assign waiter and mark as delivered
            UPDATE customer_order
            SET waiter_id = p_waiter_id,
                status = 'DELIVERED',
                delivered_at = NOW(),
                status_updated_at = NOW()
            WHERE order_id = p_order_id;

            -- Update waiter statistics
            UPDATE waiter
            SET current_table_count = current_table_count + 1,
                total_deliveries = total_deliveries + 1,
                average_delivery_time = (
                    (COALESCE(average_delivery_time, 0) * total_deliveries + v_delivery_minutes)
                    / (total_deliveries + 1)
                )
            WHERE waiter_id = p_waiter_id;
```

```
            -- Link waiter to table
            UPDATE restaurant_table
            SET assigned_waiter_id = p_waiter_id
            WHERE table_id = v_table_id;

            SET p_delivery_time_minutes = v_delivery_minutes;
            SET p_result_message = CONCAT('Success: Waiter ', p_waiter_id,
                                          ' assigned, order delivered in ',
                                          v_delivery_minutes, ' minutes.');
            COMMIT;
        END IF;
    END IF;
END$$

DELIMITER ;
```

DESAIN FUNCTION

1. Function 1: Calculate Customer Loyalty Discount

Nama Function	CalculateCustomerDiscount
Tabel dan field yang terlibat	**customer**: customer_id, total_stars **system_config**: config_key, config_value
Jenis dan Diskripsi function	**Jenis:** TVF (Table-Valued Function - Scalar Return) **Deskripsi:** Function ini digunakan untuk menghitung persentase diskon yang berhak didapat customer berdasarkan jumlah total_stars yang dimiliki. Function mengikuti business rule: ● 10-24 stars = 5% discount ● 25-49 stars = 10% discount ● 50-99 stars = 15% discount ● 100+ stars = 20% discount ● < 10 stars = 0% discount Function ini sangat penting saat cashier melakukan checkout untuk otomatis menentukan diskon yang tepat.
Input	ID customer yang akan dicek eligibilitas diskonnya

	
Output Nilai balik	**Data Type:** DECIMAL(5,2) **Format:** Persentase diskon (contoh: 5.00, 10.00, 15.00, 20.00, atau 0.00) **Contoh Output:** • Customer dengan 8 stars → return 0.00 • Customer dengan 15 stars → return 5.00 • Customer dengan 30 stars → return 10.00 • Customer dengan 75 stars → return 15.00 • Customer dengan 150 stars → return 20.00 

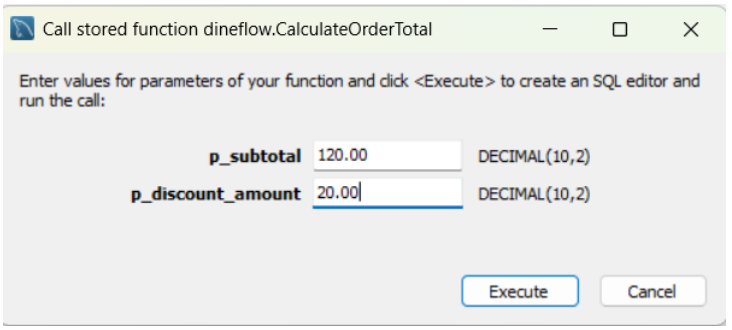
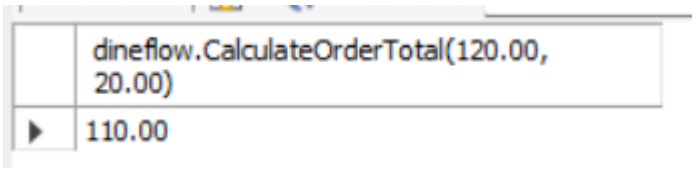
Cara akses	`SELECT CalculateCustomerDiscount(123) AS discount_percentage;`
	`SELECT`
	`c.customer_id,`
	`c.first_name,`
	`c.total_stars,`
	`CalculateCustomerDiscount(c.customer_id) AS eligible_discount`
	`FROM customer c`
	`WHERE c.customer_id = 123;`

Contoh query

```
use dineflow;
DELIMITER $$
CREATE FUNCTION CalculateCustomerDiscount(p_customer_id INT)
RETURNS DECIMAL(5,2)
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE v_total_stars INT;
    DECLARE v_discount DECIMAL(5,2);
    -- Get customer's total stars
    SELECT total_stars INTO v_total_stars
    FROM customer
    WHERE customer_id = p_customer_id;
    -- Handle NULL (customer not found)
    IF v_total_stars IS NULL THEN
        RETURN 0.00;
    END IF;
    -- Calculate discount based on star tiers
    IF v_total_stars >= 100 THEN
        SET v_discount = 20.00;
    ELSEIF v_total_stars >= 50 THEN
        SET v_discount = 15.00;
    ELSEIF v_total_stars >= 25 THEN

    ELSEIF v_total_stars >= 50 THEN
        SET v_discount = 15.00;
    ELSEIF v_total_stars >= 25 THEN
        SET v_discount = 10.00;
    ELSEIF v_total_stars >= 10 THEN
        SET v_discount = 5.00;
    ELSE
        SET v_discount = 0.00;
    END IF;

    RETURN v_discount;
END$$

DELIMITER ;
```

2. Function 2: Calculate Order Tax and Total

Nama Function	CalculateOrderTotal
Tabel dan field yang terlibat	**customer_order**: subtotal, discount_amount **system_config**: config_key ('tax_rate')

Jenis dan Diskripsi function	**Jenis:** TVF (Table-Valued Function - Scalar Return) **Deskripsi:** Function ini menghitung total akhir order berdasarkan subtotal dan discount_amount dengan menambahkan tax 10%. Formula yang digunakan: tax_amount = (subtotal - discount_amount) × 0.10 total_amount = subtotal - discount_amount + tax_amount Function ini memastikan konsistensi perhitungan tax di seluruh aplikasi dan mempermudah validasi data order.
Input	Total harga sebelum diskon dan pajak dan Jumlah diskon yang diterapkan 
Output Nilai balik	**Data Type:** DECIMAL(10,2) **Format:** Total akhir setelah diskon dan pajak 

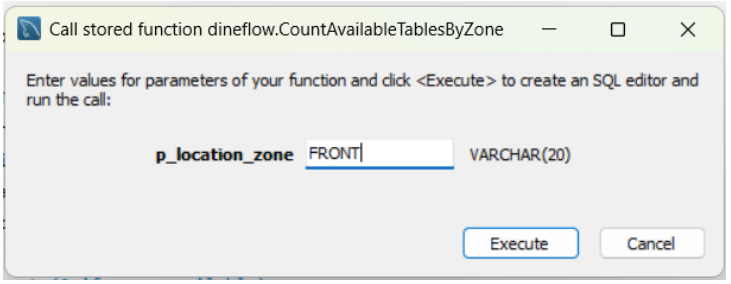
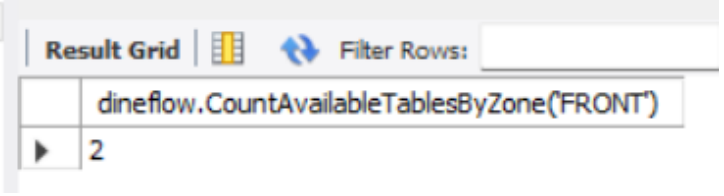
Cara akses	`SELECT CalculateOrderTotal(50.00, 5.00) AS final_total;`
	`INSERT INTO customer_order (`
	`subtotal,`
	`discount_amount,`
	`tax_amount,`
	`total_amount`
	`) VALUES (`
	`50.00,`
	`5.00,`
	`ROUND((50.00 - 5.00) * 0.10, 2),`
	`CalculateOrderTotal(50.00, 5.00)`
	`);`

Contoh query

```
use dineflow;
DELIMITER $$

CREATE FUNCTION CalculateOrderTotal(
    p_subtotal DECIMAL(10,2),
    p_discount_amount DECIMAL(10,2)
)
RETURNS DECIMAL(10,2)
DETERMINISTIC
NO SQL
BEGIN
    DECLARE v_taxable_amount DECIMAL(10,2);
    DECLARE v_tax_amount DECIMAL(10,2);
    DECLARE v_total DECIMAL(10,2);

    -- Calculate taxable amount after discount
    SET v_taxable_amount = p_subtotal - p_discount_amount;

    -- Calculate 10% tax
    SET v_tax_amount = ROUND(v_taxable_amount * 0.10, 2);

    -- Calculate final total
    SET v_total = v_taxable_amount + v_tax_amount;

    RETURN v_total;
END$$
```

3. Function 3: Count Available Tables by Zone

Nama Function	**CountAvailableTablesByZone**
Tabel dan field yang terlibat	**restaurant_table**: table_id, location_zone, is_available, is_active
Jenis dan Diskripsi function	**Jenis:** TVF (Table-Valued Function - Scalar Return) **Deskripsi:** Function ini menghitung jumlah table yang tersedia (is_available = TRUE dan is_active

	= TRUE) dalam zona lokasi tertentu. Berbeda dengan procedure, function ini return single integer value yang sangat berguna untuk: ● Quick capacity check sebelum assign table ● Dashboard display showing available tables per zone ● Decision logic dalam stored procedures ● Validation sebelum customer seating Function ini lebih efficient daripada query kompleks dan dapat digunakan dalam WHERE clauses atau SELECT statements.
Input	Zona lokasi: 'FRONT', 'MIDDLE', 'BACK', atau 'PATIO' 
Output Nilai balik	**Data Type:** INT **Format:** Jumlah table yang tersedia di zona tersebut (0 jika tidak ada) 

Cara akses	

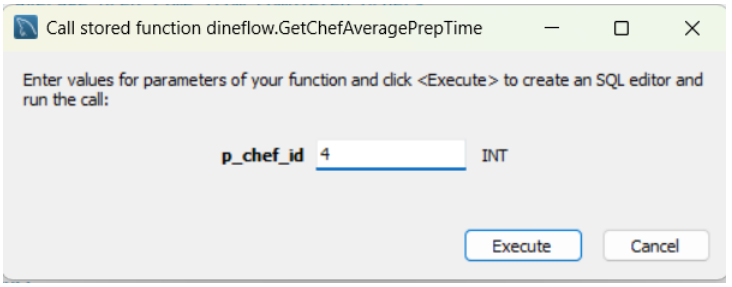
```
SELECT CountAvailableTablesByZone('FRONT') AS available_tables;
SELECT
    'FRONT' AS zone,
    CountAvailableTablesByZone('FRONT') AS available_count
UNION ALL
SELECT 'MIDDLE', CountAvailableTablesByZone('MIDDLE')
UNION ALL
SELECT 'BACK', CountAvailableTablesByZone('BACK')
UNION ALL
SELECT 'PATIO', CountAvailableTablesByZone('PATIO');
SELECT * FROM restaurant_table
WHERE location_zone = 'FRONT'
  AND CountAvailableTablesByZone('FRONT') > 0;
IF CountAvailableTablesByZone('PATIO') = 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'No tables available in PATIO zone';
END IF;
SELECT
    location_zone,
    COUNT(*) AS total_tables,
    CountAvailableTablesByZone(location_zone) AS available_tables,
    COUNT(*) - CountAvailableTablesByZone(location_zone) AS occupied_tables
FROM restaurant_table
WHERE is_active = TRUE
GROUP BY location_zone;
```

Contoh query:

```
use dineflow;
DELIMITER $

CREATE FUNCTION CountAvailableTablesByZone(p_location_zone VARCHAR(20))
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE v_count INT;

    SELECT COUNT(*) INTO v_count
    FROM restaurant_table
    WHERE location_zone = p_location_zone
      AND is_available = TRUE
      AND is_active = TRUE;

    -- Return count (0 if none available)
    RETURN IFNULL(v_count, 0);
END$

DELIMITER ;
```

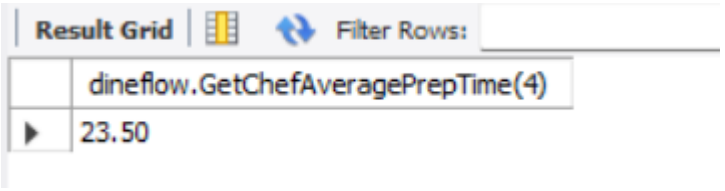
4. Function 4: Calculate Chef Average Preparation Time

Nama Function	GetChefAveragePrepTime

Tabel dan field yang terlibat	**customer_order**: chef_id, actual_prep_time, status **chef**: chef_id **employee**: employee_id, first_name, last_name
Jenis dan Diskripsi function	**Jenis:** TVF (Table-Valued Function - Scalar Return) **Deskripsi:** Function ini menghitung rata-rata waktu persiapan (dalam menit) untuk seorang chef berdasarkan semua order yang telah completed. Hanya menghitung order dengan status 'COMPLETED' atau 'DELIVERED' dan memiliki actual_prep_time yang tidak NULL. Function ini digunakan untuk performance evaluation chef dan identifikasi bottleneck di kitchen operations.
Input	ID chef yang akan dihitung average prep time-nya 
Output Nilai balik	**Data Type:** DECIMAL(5,2) **Format:** Rata-rata waktu dalam menit (contoh: 16.45 berarti 16 menit 27 detik) **Interpretasi Output:** • < 15.00 = Excellent performance (di bawah target) • 15.00 - 20.00 = Good performance (dalam range normal) • 20.00 - 25.00 = Needs attention (mendekati warning threshold) • 25.00 = Poor performance (sering exceed warning)

<table>
<tr><td></td><td>

- NULL = Chef belum complete order apapun



</td></tr>
<tr><td>Cara akses</td><td>

```sql
SELECT GetChefAveragePrepTime(5) AS avg_prep_minutes;
SELECT
    e.employee_id,
    CONCAT(e.first_name, ' ', e.last_name) AS chef_name,
    ch.specialty,
    GetChefAveragePrepTime(ch.chef_id) AS avg_prep_time,
    ch.total_orders_prepared,
    ch.orders_exceeding_sla
FROM chef ch
JOIN employee e ON ch.chef_id = e.employee_id
WHERE e.employment_status = 'ACTIVE'
ORDER BY avg_prep_time;
```

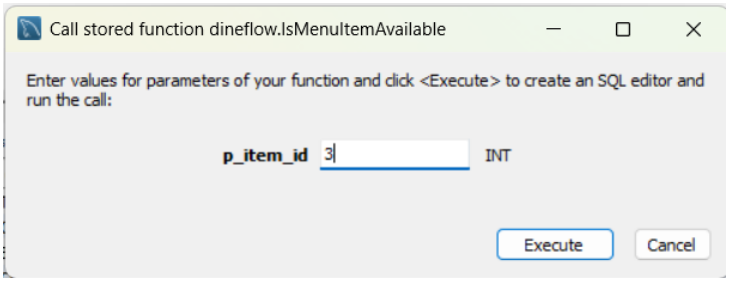
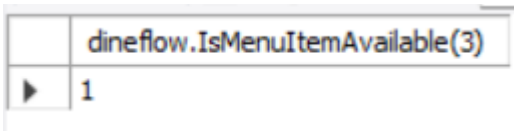
</td></tr>
</table>

Contoh query:

```sql
CREATE FUNCTION GetChefAveragePrepTime(p_chef_id INT)
RETURNS DECIMAL(5,2)
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE v_avg_time DECIMAL(5,2);

    -- Calculate average prep time from completed orders
    SELECT AVG(actual_prep_time) INTO v_avg_time
    FROM customer_order
    WHERE chef_id = p_chef_id
      AND status IN ('COMPLETED', 'DELIVERED')
      AND actual_prep_time IS NOT NULL;

    -- Return NULL if no completed orders
    IF v_avg_time IS NULL THEN
        RETURN NULL;
    END IF;

    RETURN v_avg_time;
END$$

DELIMITER ;
```

5. Function 5: Check Menu Item Availability

Nama Function	IsMenuItemAvailable
Tabel dan field yang terlibat	**menu_item**: item_id, is_available, is_active
Jenis dan Diskripsi function	**Jenis:** TVF (Table-Valued Function - Scalar Return) **Deskripsi:** Function ini melakukan pengecekan sederhana apakah menu item dapat dipesan atau tidak. Return TRUE (1) jika item tersedia (is_available = TRUE AND is_active = TRUE), return FALSE (0) jika tidak tersedia atau tidak aktif. Function ini digunakan oleh cashier application sebelum menambahkan item ke order untuk mencegah order item yang sedang unavailable.

Input	ID menu item yang akan dicek ketersediaannya 
Output Nilai balik	**Data Type:** BOOLEAN (TINYINT(1)) **Format:** • 1 (TRUE) = Item tersedia dan dapat dipesan • 0 (FALSE) = Item tidak tersedia atau sudah dihapus 

Cara akses	`SELECT IsMenuItemAvailable(15) AS can_order;`
	`SELECT`
	`  m.item_id,`
	`  m.item_name,`
	`  m.category,`
	`  m.price,`
	`  IsMenuItemAvailable(m.item_id) AS is_orderable,`
	`  CASE`
	`  WHEN IsMenuItemAvailable(m.item_id) = 1 THEN 'Available'`
	`  ELSE 'Not Available'`
	`  END AS status_label`
	`FROM menu_item m`
	`WHERE m.category = 'MAIN';`
	`IF IsMenuItemAvailable(item_id) = 1 THEN`
	`ELSE`
	`END IF;`

Contoh query:

```
CREATE FUNCTION IsMenuItemAvailable(p_item_id INT)
RETURNS BOOLEAN
DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE v_is_available BOOLEAN;
    DECLARE v_is_active BOOLEAN;
    -- Get item availability status
    SELECT is_available, is_active INTO v_is_available, v_is_active
    FROM menu_item
    WHERE item_id = p_item_id;
    -- Return FALSE if item not found
    IF v_is_available IS NULL THEN
        RETURN FALSE;
    END IF;
    -- Return TRUE only if both available and active
    IF v_is_available = TRUE AND v_is_active = TRUE THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END$$
DELIMITER ;
```

DESAIN TRIGGER

A. Trigger 1: Update Customer Stars After Order Payment

trg_after_transaction_insert_update_stars

T (Tentukan)	**Tabel yang Terlibat:** • **payment_transaction** (trigger table) • **customer** (updated table) • **loyalty_star_transaction** (audit table) **Field yang Digunakan:**

	• payment_transaction: transaction_id, order_id, transaction_status • customer: customer_id, total_stars • customer_order: customer_id, stars_earned • loyalty_star_transaction: star_amount, balance_before, balance_after
K (Kapan)	**Event:** AFTER INSERT **Timing:** Setelah record baru diinsert ke tabel payment_transaction **Kondisi:** Hanya fire jika transaction_status = 'COMPLETED' dan order memiliki customer_id (bukan guest) **Skenario Aktivasi:** 1. Cashier memproses pembayaran order 2. Record transaction diinsert dengan status COMPLETED 3. Trigger otomatis credit 1 star ke customer account 4. Audit trail dicatat di loyalty_star_transaction
P (Perintahnya)	**Aksi yang Dilakukan:** 1. Ambil customer_id dari order yang terkait dengan transaction 2. Cek apakah customer_id NOT NULL (bukan guest order) 3. Ambil current total_stars dari customer (balance_before) 4. UPDATE customer table: total_stars = total_stars + 1 5. INSERT ke loyalty_star_transaction untuk audit trail: ○ transaction_type = 'EARNED' ○ star_amount = +1 ○ balance_before = old value ○ balance_after = old value + 1 6. UPDATE customer: is_vip = TRUE jika total_stars >= 100

Business Rule yang Di-enforce:

- BR-010: 1 star earned per completed order
- BR-018: Stars hanya credited untuk paying customers (bukan guest)

```
DELIMITER $$
CREATE TRIGGER trg_after_transaction_insert_update_stars
AFTER INSERT ON payment_transaction
FOR EACH ROW
BEGIN
    DECLARE v_customer_id INT;
    DECLARE v_current_stars INT;
    DECLARE v_stars_to_add INT;

    IF NEW.transaction_status = 'COMPLETED' THEN

        SELECT customer_id, stars_earned
        INTO v_customer_id, v_stars_to_add
        FROM customer_order
        WHERE order_id = NEW.order_id;

        IF v_customer_id IS NOT NULL THEN

            -- Get current star balance
            SELECT total_stars INTO v_current_stars
            FROM customer
            WHERE customer_id = v_customer_id;

            UPDATE customer
            SET total_stars = total_stars + v_stars_to_add,
                last_order_date = CURRENT_DATE,
                total_lifetime_orders = total_lifetime_orders + 1,
                is_vip = IF(total_stars + v_stars_to_add >= 100, TRUE, FALSE)


           WHERE customer_id = v_customer_id;

           INSERT INTO loyalty_star_transaction (
               customer_id,
               transaction_type,
               star_amount,
               balance_before,
               balance_after,
               order_id,
               transaction_id,
               transaction_date,
               transaction_time
           ) VALUES (
               v_customer_id,
               'EARNED',
               v_stars_to_add,
               v_current_stars,
               v_current_stars + v_stars_to_add,
               NEW.order_id,
               NEW.transaction_id,
               NEW.transaction_date,
               NEW.transaction_time
           );
       END IF;
   END IF;
END$$

DELIMITER ;
```

B. Trigger 2: Reverse Stars When Order Refunded

trg_after_refund_approved_reverse_stars

T (Tentukan)	**Tabel yang Terlibat:** • **refund** (trigger table) • **customer** (updated table) • **loyalty_star_transaction** (audit table) • **customer_order** (reference table) **Field yang Digunakan:** • refund: refund_id, order_id, approval_status, stars_reversed • customer: customer_id, total_stars • customer_order: customer_id, stars_earned
K (Kapan)	**Event:** AFTER UPDATE **Timing:** Setelah refund record di-update **Kondisi:** Hanya fire jika approval_status berubah dari 'PENDING' ke 'APPROVED' **Skenario Aktivasi:** 1. Manager approve refund request 2. Refund status berubah dari PENDING → APPROVED 3. Trigger otomatis mengurangi stars dari customer 4. Audit trail dicatat sebagai 'REVERSED'
P (Perintahnya)	**Aksi yang Dilakukan:** 1. Cek apakah NEW.approval_status = 'APPROVED' AND OLD.approval_status = 'PENDING' 2. Ambil customer_id dari order yang di-refund 3. Ambil jumlah stars yang harus di-reverse dari order.stars_earned 4. Cek apakah customer_id NOT NULL 5. Ambil current total_stars (balance_before)

6. UPDATE customer: total_stars = total_stars - stars_reversed (minimum 0)
7. INSERT audit ke loyalty_star_transaction:
    - transaction_type = 'REVERSED'
    - star_amount = negative value
    - refund_id = linked
8. UPDATE customer: is_vip = FALSE jika total_stars < 100

Business Rule yang Di-enforce:

- BR-018: Refunded orders do not earn stars; stars removed if already credited
- BR-030: Stars removed when order refunded

```
DELIMITER $$
CREATE TRIGGER trg_after_refund_approved_reverse_stars
AFTER UPDATE ON refund
FOR EACH ROW
BEGIN
    DECLARE v_customer_id INT;
    DECLARE v_current_stars INT;
    DECLARE v_stars_to_reverse INT;

    -- Only process when refund is approved
    IF NEW.approval_status = 'APPROVED' AND OLD.approval_status = 'PENDING' THEN

        -- Get customer_id and stars from the refunded order
        SELECT customer_id, stars_earned
        INTO v_customer_id, v_stars_to_reverse
        FROM customer_order
        WHERE order_id = NEW.order_id;

        -- Only process if customer exists
        IF v_customer_id IS NOT NULL AND v_stars_to_reverse > 0 THEN

            -- Get current star balance
            SELECT total_stars INTO v_current_stars
            FROM customer
            WHERE customer_id = v_customer_id;
```

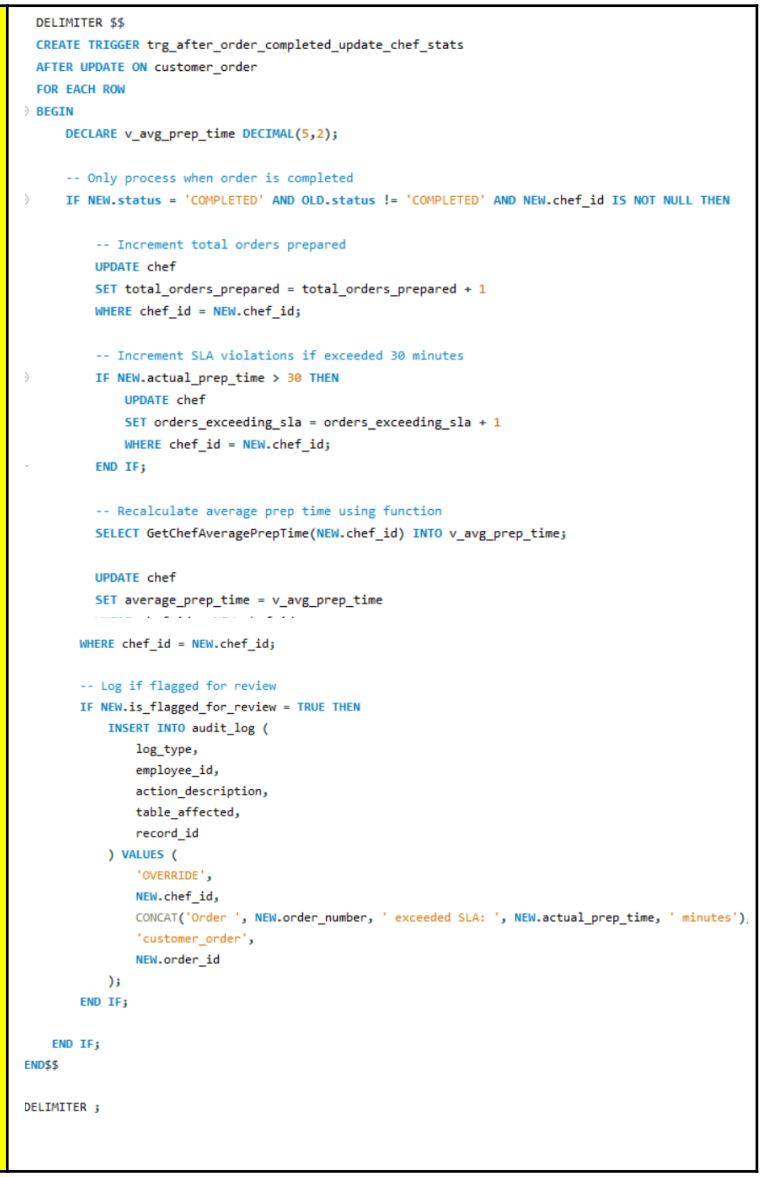
```
        UPDATE customer
        SET total_stars = GREATEST(0, total_stars - v_stars_to_reverse),
            is_vip = IF(GREATEST(0, total_stars - v_stars_to_reverse) >= 100, TRUE, FALSE)
        WHERE customer_id = v_customer_id;

        -- Create audit record for reversal
        INSERT INTO loyalty_star_transaction (
            customer_id,
            transaction_type,
            star_amount,
            balance_before,
            balance_after,
            order_id,
            refund_id,
            transaction_date,
            transaction_time,
            notes
        ) VALUES (
            v_customer_id,
            'REVERSED',
            -v_stars_to_reverse,
            v_current_stars,
            GREATEST(0, v_current_stars - v_stars_to_reverse),
            NEW.order_id,
            NEW.refund_id,
            NEW.refund_date,
            NEW.refund_time,
                CONCAT('Stars reversed due to refund: ', NEW.refund_reason)
            );

            UPDATE refund
            SET stars_reversed = v_stars_to_reverse
            WHERE refund_id = NEW.refund_id;

        END IF;
    END IF;
END$$

DELIMITER ;
```

C. Trigger 3: Update Chef Performance Metrics

trg_after_order_completed_update_chef_stats

T (Tentukan)	**Tabel yang Terlibat:** • **customer_order** (trigger table) • **chef** (updated table) **Field yang Digunakan:** • customer_order: order_id, chef_id, status, actual_prep_time, is_flagged_for_review • chef: chef_id, total_orders_prepared, average_prep_time, orders_exceeding_sla

K (Kapan)	**Event:** AFTER UPDATE **Timing:** Setelah order status di-update **Kondisi:** Hanya fire jika status berubah menjadi 'COMPLETED' dan chef_id NOT NULL **Skenario Aktivasi:** 1. Chef marks order as completed in kitchen display 2. Order status berubah dari 'IN_PROCESS' → 'COMPLETED' 3. Trigger otomatis update chef performance metrics 4. Increment total_orders_prepared, recalculate average_prep_time
P (Perintahnya)	**Aksi yang Dilakukan:** 1. Cek apakah NEW.status = 'COMPLETED' AND OLD.status != 'COMPLETED' 2. Cek apakah NEW.chef_id NOT NULL 3. Increment chef.total_orders_prepared by 1 4. Recalculate chef.average_prep_time menggunakan function GetChefAveragePrepTime() 5. Jika actual_prep_time > 30 minutes, increment chef.orders_exceeding_sla 6. Jika order is_flagged_for_review = TRUE, log in audit **Business Rule yang Di-enforce:** ● BR-004: Maximum 30 minutes from order creation to completion (SLA) ● Performance tracking untuk chef evaluation

```
DELIMITER $$
CREATE TRIGGER trg_after_order_completed_update_chef_stats
AFTER UPDATE ON customer_order
FOR EACH ROW
BEGIN
    DECLARE v_avg_prep_time DECIMAL(5,2);

    -- Only process when order is completed
    IF NEW.status = 'COMPLETED' AND OLD.status != 'COMPLETED' AND NEW.chef_id IS NOT NULL THEN

        -- Increment total orders prepared
        UPDATE chef
        SET total_orders_prepared = total_orders_prepared + 1
        WHERE chef_id = NEW.chef_id;

        -- Increment SLA violations if exceeded 30 minutes
        IF NEW.actual_prep_time > 30 THEN
            UPDATE chef
            SET orders_exceeding_sla = orders_exceeding_sla + 1
            WHERE chef_id = NEW.chef_id;
        END IF;

        -- Recalculate average prep time using function
        SELECT GetChefAveragePrepTime(NEW.chef_id) INTO v_avg_prep_time;

        UPDATE chef
        SET average_prep_time = v_avg_prep_time
      WHERE chef_id = NEW.chef_id;

      -- Log if flagged for review
      IF NEW.is_flagged_for_review = TRUE THEN
          INSERT INTO audit_log (
              log_type,
              employee_id,
              action_description,
              table_affected,
              record_id
          ) VALUES (
              'OVERRIDE',
              NEW.chef_id,
              CONCAT('Order ', NEW.order_number, ' exceeded SLA: ', NEW.actual_prep_time, ' minutes'),
              'customer_order',
              NEW.order_id
          );
      END IF;

    END IF;
END$$

DELIMITER ;
```

D. Trigger 4: Update Cashier Performance Metrics.

trg_after_order_queued_update_cashier_stats

T (Tentukan)	**Tabel yang Terlibat:** • **customer_order** (trigger table) • **cashier** (updated table) **Field yang Digunakan:** • customer_order: order_id, cashier_id, status, created_at, queued_at

	• cashier: cashier_id, total_orders_processed, average_order_processing_time
K (Kapan)	**Event:** AFTER UPDATE **Timing:** Setelah order status di-update **Kondisi:** Hanya fire jika status berubah dari 'PENDING' ke 'QUEUED' (payment completed, sent to kitchen) **Skenario Aktivasi:** 1. Cashier completes payment processing 2. Order status berubah dari 'PENDING' → 'QUEUED' 3. Trigger otomatis update cashier performance metrics 4. Calculate time dari order creation sampai queued (processing time)
P (Perintahnya)	**Aksi yang Dilakukan:** 1. Cek apakah NEW.status = 'QUEUED' AND OLD.status = 'PENDING' 2. Cek apakah NEW.cashier_id NOT NULL 3. Calculate processing time: TIMESTAMPDIFF(SECOND, created_at, queued_at) / 60 4. Increment cashier.total_orders_processed by 1 5. Recalculate cashier.average_order_processing_time menggunakan running average formula: new_avg = ((old_avg × old_count) + new_time) / new_count **Business Rule yang Di-enforce:** • NFR-U-001: Cashier order entry efficiency (target ≤ 5 clicks/minutes) • Performance tracking untuk staff evaluation

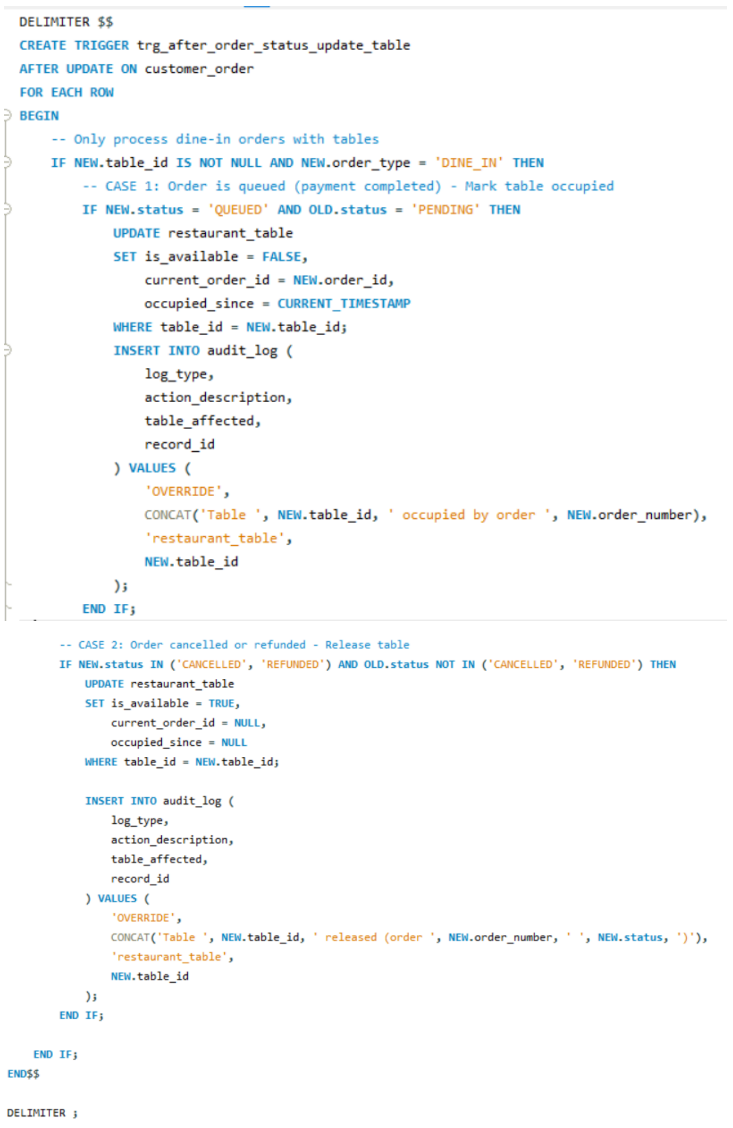
```
DELIMITER $$

CREATE TRIGGER trg_after_order_queued_update_cashier_stats
AFTER UPDATE ON customer_order
FOR EACH ROW
BEGIN
    DECLARE v_processing_time DECIMAL(5,2);
    DECLARE v_current_count INT;
    DECLARE v_current_avg DECIMAL(5,2);
    DECLARE v_new_avg DECIMAL(5,2);

    -- Only process when order moves from PENDING to QUEUED
    IF NEW.status = 'QUEUED' AND OLD.status = 'PENDING' AND NEW.cashier_id IS NOT NULL THEN

        -- Calculate processing time in minutes
        SET v_processing_time = TIMESTAMPDIFF(SECOND, NEW.created_at, NEW.queued_at) / 60.0;

        -- Get current cashier stats
        SELECT total_orders_processed, average_order_processing_time
        INTO v_current_count, v_current_avg
        FROM cashier
        WHERE cashier_id = NEW.cashier_id;

        -- Handle NULL average (first order)
        IF v_current_avg IS NULL THEN
            SET v_current_avg = 0;
        END IF;

        SET v_new_avg = ((v_current_avg * v_current_count) + v_processing_time) / (v_current_count + 1);

        -- Update cashier performance metrics
        UPDATE cashier
        SET total_orders_processed = total_orders_processed + 1,
            average_order_processing_time = v_new_avg,
            last_shift_date = CURRENT_DATE
        WHERE cashier_id = NEW.cashier_id;

    END IF;
END$$

DELIMITER ;
```

E. Trigger 5: Auto-Update Table Availability Status.

trg_after_order_status_update_table

T (Tentukan)	**Tabel yang Terlibat:** ● **customer_order** (trigger table) ● **restaurant_table** (updated table) **Field yang Digunakan:** ● customer_order: order_id, table_id, status, order_type ● restaurant_table: table_id, is_available, current_order_id, occupied_since

K (Kapan)	**Event:** AFTER UPDATE **Timing:** Setelah order status di-update **Kondisi:** Fire pada status change yang mempengaruhi table availability: ● Status → 'QUEUED': Mark table as occupied ● Status → 'DELIVERED': Keep table occupied (customer eating) ● Status → 'CANCELLED' or 'REFUNDED': Release table **Skenario Aktivasi:** 1. Dine-in order created and paid → table marked occupied 2. Order delivered → table stays occupied 3. Waiter marks table available setelah customer leaves 4. Order cancelled → auto-release table
P (Perintahnya)	**Aksi yang Dilakukan:** 1. Cek apakah order is DINE_IN (table_id NOT NULL) 2. **When status = 'QUEUED':** ○ UPDATE table: is_available = FALSE ○ SET current_order_id = order_id ○ SET occupied_since = CURRENT_TIMESTAMP 3. **When status = 'CANCELLED' or 'REFUNDED':** ○ UPDATE table: is_available = TRUE ○ SET current_order_id = NULL ○ SET occupied_since = NULL 4. Log table status changes in audit_log **Business Rule yang Di-enforce:** ● BR-035: One order per table at a time ● BR-036: Tables remain occupied until marked available

	• Automatic table lifecycle management 

```
DELIMITER $$
CREATE TRIGGER trg_after_order_status_update_table
AFTER UPDATE ON customer_order
FOR EACH ROW
BEGIN
    -- Only process dine-in orders with tables
    IF NEW.table_id IS NOT NULL AND NEW.order_type = 'DINE_IN' THEN
        -- CASE 1: Order is queued (payment completed) - Mark table occupied
        IF NEW.status = 'QUEUED' AND OLD.status = 'PENDING' THEN
            UPDATE restaurant_table
            SET is_available = FALSE,
                current_order_id = NEW.order_id,
                occupied_since = CURRENT_TIMESTAMP
            WHERE table_id = NEW.table_id;
            INSERT INTO audit_log (
                log_type,
                action_description,
                table_affected,
                record_id
            ) VALUES (
                'OVERRIDE',
                CONCAT('Table ', NEW.table_id, ' occupied by order ', NEW.order_number),
                'restaurant_table',
                NEW.table_id
            );
        END IF;

        -- CASE 2: Order cancelled or refunded - Release table
        IF NEW.status IN ('CANCELLED', 'REFUNDED') AND OLD.status NOT IN ('CANCELLED', 'REFUNDED') THEN
            UPDATE restaurant_table
            SET is_available = TRUE,
                current_order_id = NULL,
                occupied_since = NULL
            WHERE table_id = NEW.table_id;

            INSERT INTO audit_log (
                log_type,
                action_description,
                table_affected,
                record_id
            ) VALUES (
                'OVERRIDE',
                CONCAT('Table ', NEW.table_id, ' released (order ', NEW.order_number, ' ', NEW.status, ')'),
                'restaurant_table',
                NEW.table_id
            );
        END IF;

    END IF;
END$$

DELIMITER ;
```

F. Trigger 6: Update Waiter Table Count

trg_after_table_assignment_update_waiter_count

T (Tentukan)	**Tabel yang Terlibat:** • **restaurant_table** (trigger table) • **waiter** (updated table) **Field yang Digunakan:**

	• restaurant_table: table_id, assigned_waiter_id • waiter: waiter_id, current_table_count, max_tables
K (Kapan)	**Event:** AFTER UPDATE **Timing:** Setelah waiter assignment di table berubah **Kondisi:** Fire ketika assigned_waiter_id berubah (assigned, reassigned, atau unassigned) **Skenario Aktivasi:** 1. Manager/system assigns waiter ke table → increment waiter's count 2. Table reassigned ke waiter lain → decrement old, increment new 3. Table unassigned (waiter goes off-duty) → decrement count 4. Automatic validation: prevent exceeding max_tables limit
P (Perintahnya)	**Aksi yang Dilakukan:** 1. **When waiter assigned (OLD.assigned_waiter_id IS NULL, NEW NOT NULL):** ○ Increment NEW waiter's current_table_count ○ Validate: current_table_count <= max_tables 2. **When waiter unassigned (OLD NOT NULL, NEW IS NULL):** ○ Decrement OLD waiter's current_table_count 3. **When waiter reassigned (OLD != NEW, both NOT NULL):** ○ Decrement OLD waiter's count ○ Increment NEW waiter's count 4. Prevent assignment if waiter at max capacity

Business Rule yang Di-enforce:

- BR-037: Each waiter assigned maximum 6 tables per shift
- Real-time workload tracking
- Automatic capacity management

```
DELIMITER $$

CREATE TRIGGER trg_after_table_assignment_update_waiter_count
AFTER UPDATE ON restaurant_table
FOR EACH ROW
BEGIN
    DECLARE v_max_tables INT;
    DECLARE v_current_count INT;

    -- CASE 1: New waiter assigned (was NULL, now has waiter)
    IF OLD.assigned_waiter_id IS NULL AND NEW.assigned_waiter_id IS NOT NULL THEN

        -- Get waiter's current count and max
        SELECT current_table_count, max_tables
        INTO v_current_count, v_max_tables
        FROM waiter
        WHERE waiter_id = NEW.assigned_waiter_id;

        -- Validate capacity (should be checked in application, but enforce here too)
        IF v_current_count >= v_max_tables THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Waiter has reached maximum table capacity';
        END IF;

        -- Increment waiter's table count
        UPDATE waiter
```

```
      AND NEW.assigned_waiter_id IS NOT NULL
      AND OLD.assigned_waiter_id != NEW.assigned_waiter_id THEN

        -- Decrement old waiter's count
        UPDATE waiter
        SET current_table_count = GREATEST(0, current_table_count - 1)
        WHERE waiter_id = OLD.assigned_waiter_id;

        -- Get new waiter's capacity
        SELECT current_table_count, max_tables
        INTO v_current_count, v_max_tables
        FROM waiter
        WHERE waiter_id = NEW.assigned_waiter_id;

        -- Validate capacity
        IF v_current_count >= v_max_tables THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'New waiter has reached maximum table capacity';
        END IF;

        -- Increment new waiter's count
        UPDATE waiter
        SET current_table_count = current_table_count + 1
        WHERE waiter_id = NEW.assigned_waiter_id;

    END IF;
END$$
DELIMITER ;
```

DESAIN HIGH AVAILABILITY (HA)

Analisis Kebutuhan high Availability

Operasional Bisnis Kritis:

- Pemrosesan pesanan (*order processing*): kasir tidak dapat menerima pembayaran
- Tampilan dapur (*kitchen display*): koki tidak dapat melihat pesanan
- Manajemen meja (*table management*): pelayan tidak dapat melacak pengantaran
- Pelacakan loyalitas pelanggan (*customer loyalty tracking*): poin tidak tercatat
- Dan Juga ada dampak financial

Kebutuhan Waktu Operasional dan Target *Uptime*

Jam Operasional Bisnis:

- *10.00 – 22.00 WIB* (12 jam per hari)
  Jam operasional ini dipilih karena seluruh aktivitas transaksi, layanan pelanggan, dan operasional dapur hanya berlangsung pada rentang waktu tersebut.

Target *Uptime*:

- *99,5%* selama jam operasional
  Target ini dipilih untuk menyeimbangkan antara kebutuhan keandalan sistem dan keterbatasan sumber daya, karena sistem digunakan secara intensif hanya saat jam bisnis aktif.

Batas *Downtime* yang Dapat Diterima:

- Per hari: ± *3,6 menit*
- Per bulan: ± *1,8 jam*
- Per tahun: ± *21,6 jam* (sebagian besar di luar jam operasional)
  Batas ini ditetapkan agar pemeliharaan dan gangguan minor tidak berdampak langsung pada proses pelayanan pelanggan.

Skenario yang Tidak Dapat Diterima:

- *Database* tidak tersedia saat jam makan siang (*12.00–14.00*)
- *Database* tidak tersedia saat jam makan malam (*18.00–20.00*)
- Kehilangan data pesanan yang telah selesai
- Hilangnya poin loyalitas pelanggan (*customer loyalty stars*)

Arsitektur yang Dipilih: Replikasi MySQL *Master–Slave* dengan *Auto-Failover*

Jenis Arsitektur:

- *Active–Passive* dengan *automatic failover*

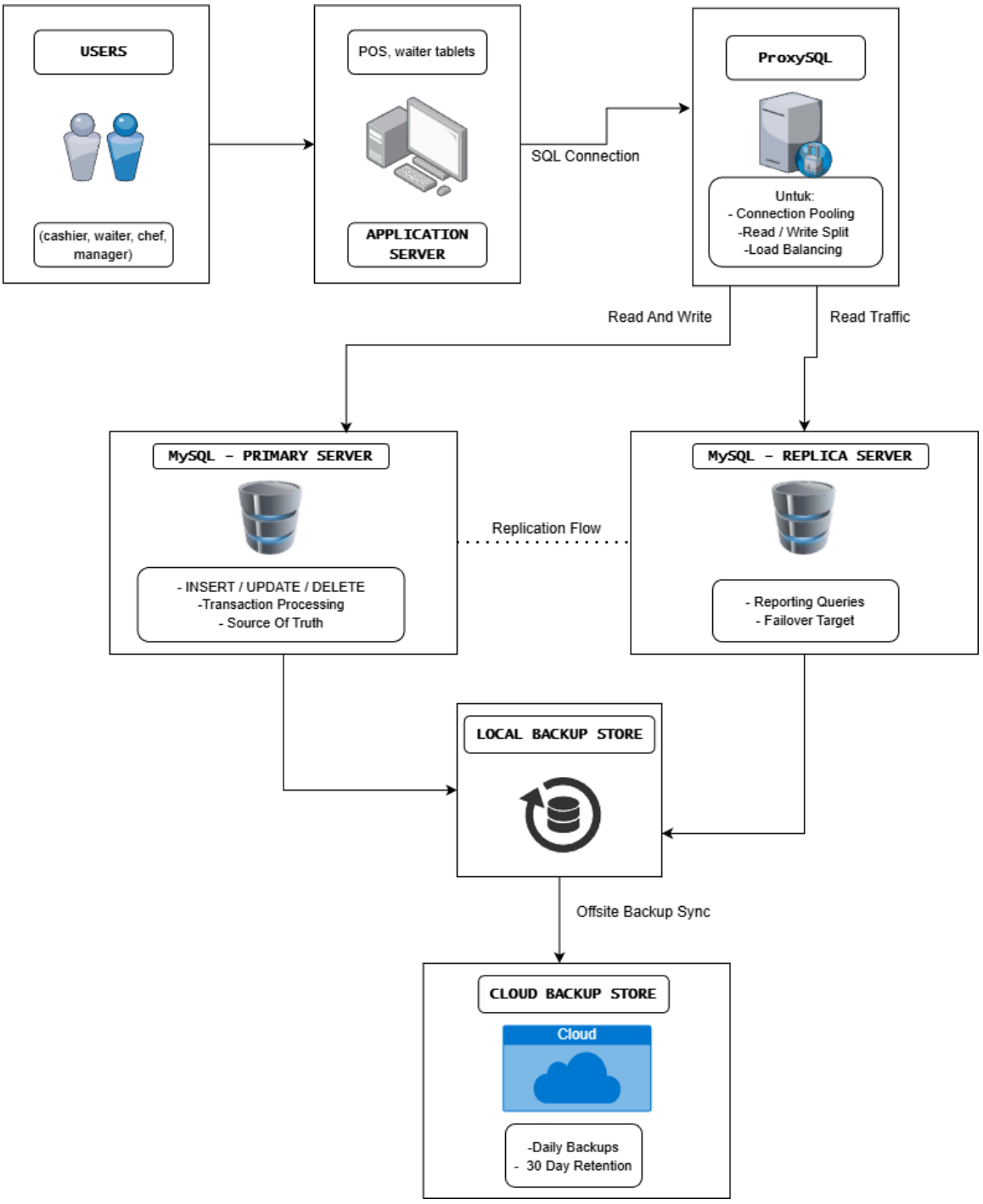
Alasan Pemilihan Arsitektur:

1. **Efisien dari Segi Biaya (*Cost-Effective*):**
   Hanya membutuhkan dua server basis data, sehingga biaya implementasi dan pemeliharaan tetap rendah tanpa mengorbankan keandalan sistem.
2. **Mudah Diimplementasikan (*Simple to Implement*):**
   Menggunakan fitur replikasi bawaan MySQL, sehingga tidak memerlukan konfigurasi yang kompleks atau teknologi tambahan.
3. **Teknologi Teruji (*Proven Technology*):**
   Model replikasi ini telah digunakan secara luas pada berbagai sistem produksi dan terbukti stabil dalam jangka panjang.
4. **Perpindahan Cepat saat Gangguan (*Fast Failover*):**
   Dengan pemantauan yang tepat, perpindahan dari *master* ke *slave* dapat dilakukan dalam waktu kurang dari *2 menit*, sehingga *downtime* dapat diminimalkan.
5. **Skalabilitas Baca (*Read Scalability*):**
   Server *slave* dapat digunakan untuk kebutuhan laporan dan *query* baca, sehingga beban pada *master* berkurang.

Arsitektur yang Tidak Dipilih dan Alasannya:

- **Single Server:** Tidak memiliki redundansi dan menjadi *single point of failure*.
- **Multi-Master:** Terlalu kompleks untuk skala restoran dan berisiko menimbulkan konflik data.
- **MySQL Cluster (NDB):** Berlebihan untuk kebutuhan sistem ini, dengan biaya dan kompleksitas yang tinggi.
- **Cloud-Only:** Sistem restoran membutuhkan solusi *on-premise* untuk menjaga keandalan jaringan dan operasional lokal.

Desain Arsitektur



Deskripsi Komponen Sistem

1. Server *Master Database* (Primary)

Peran:

- Menangani seluruh operasi tulis (*write operations*) dan sebagian besar operasi baca (*read operations*).
  Peran ini dipilih untuk menjaga konsistensi data karena hanya satu server yang bertanggung jawab atas perubahan data.

Spesifikasi dan Alasan Pemilihan:

Perangkat Keras (*Hardware*):

- **Dell PowerEdge T340 atau setara**
  Dipilih karena merupakan server kelas *entry-level enterprise* yang stabil dan umum digunakan untuk sistem produksi skala menengah.
- **CPU: Intel Xeon E-2234 (4 core, 3.6 GHz)**
  Cukup untuk menangani transaksi paralel restoran tanpa biaya berlebihan.
- **RAM: 32 GB DDR4 ECC**
  ECC digunakan untuk mencegah kesalahan memori (*memory corruption*), penting untuk sistem basis data.
- **Penyimpanan: 2×500 GB SSD RAID 1 (*mirrored*)**
  SSD memberikan kecepatan tinggi, sementara RAID 1 dipilih untuk redundansi data jika salah satu disk gagal.
- **Jaringan: 1 Gbps Ethernet**
  Memadai untuk lalu lintas data internal antara aplikasi, *proxy*, dan server replika.

Perangkat Lunak (*Software*):

- *Ubuntu Server 22.04 LTS + MySQL 8.0*
  Dipilih karena stabil, mendapat dukungan jangka panjang (*LTS*), dan kompatibel penuh dengan replikasi MySQL.

Alamat IP:

- *192.168.1.10* (statis, jaringan internal)
  IP statis digunakan untuk menjaga konsistensi konfigurasi replikasi dan *failover*.

2. Server *Slave Database* (Standby)

Peran:

- Menyimpan salinan data *real-time* dari *Master* dan siap mengambil alih saat terjadi kegagalan (*automatic failover*).

Alasan Spesifikasi Identik dengan Master:

- Spesifikasi yang sama memastikan performa tetap konsisten setelah *failover*, tanpa penurunan kecepatan atau kapasitas sistem.

Spesifikasi:

- **Perangkat keras:** Sama dengan *Master* (Dell PowerEdge T340)
- **Perangkat lunak:** *Ubuntu Server 22.04 LTS + MySQL 8.0*

Alamat IP:

- *192.168.1.11* (statis, jaringan internal)

3. ProxySQL (*Connection Router & Failover Manager*)

Peran:

- Bertindak sebagai *proxy* cerdas yang mengatur koneksi, melakukan *query routing*, dan menangani *automatic failover*.

Alasan Pemilihan ProxySQL:

- Deteksi *failover* otomatis (< *2 menit*), sesuai dengan target *uptime*.
- *Connection pooling* meningkatkan performa dengan mengurangi beban koneksi ke database.
- *Query routing* memisahkan *read* ke *slave* dan *write* ke *master*.
- Transparan bagi aplikasi, sehingga tidak memerlukan perubahan kode.

Spesifikasi:

- **Instalasi:** Server terpisah ringan atau digabung dengan server aplikasi
- **Perangkat keras:** 4 CPU core, 8 GB RAM
  Spesifikasi ini dipilih karena ProxySQL tidak menyimpan data, hanya memproses lalu lintas koneksi.

Alamat IP:

- *192.168.1.20*
  Seluruh aplikasi terhubung ke alamat ini untuk menyederhanakan manajemen koneksi.

4. Sistem Monitoring

Tujuan:

- Mendeteksi gangguan sistem secara dini dan memberikan notifikasi cepat kepada pihak terkait.

Alat yang Digunakan dan Alasannya:

1. **Prometheus + Grafana**
   Digunakan untuk visualisasi metrik secara *real-time* melalui *dashboard* yang mudah dipahami.
2. **MySQL Exporter**
   Mengumpulkan metrik khusus MySQL yang tidak tersedia secara default.
3. **Alertmanager**
   Mengelola dan mengirim notifikasi saat ambang batas (*threshold*) terlampaui.

Metrik Kunci yang Dipantau:

- *Replication lag* Master–Slave (< *5 detik* masih dapat diterima)
- Jumlah koneksi database (peringatan jika > *150*)
- Penggunaan disk (peringatan pada *80%*)
- Penggunaan CPU/RAM (peringatan pada *90%*)
- Jumlah *failed queries* per menit
- Keberhasilan atau kegagalan proses *backup*

Saluran Notifikasi (*Alert Channels*):

- SMS ke manajer restoran (untuk respons cepat)
- Email ke tim IT (untuk penanganan teknis)
- Notifikasi *Slack/Discord* ke tim operasional

Penjelasan Skenario *Failover*

1. Kegagalan *Master Database*

Skenario:
Master database tidak dapat diakses akibat kegagalan perangkat keras, pemadaman listrik, atau *system crash*.

Yang Terjadi:

- ProxySQL secara terus-menerus memeriksa kondisi *master database*.
- Ketika *master* tidak merespons, ProxySQL menandainya sebagai *offline*.
- *Slave database* secara otomatis dipromosikan menjadi *master* baru.
- Seluruh lalu lintas database dialihkan ke server yang telah dipromosikan.

Alasan Mekanisme Ini Efektif:

- *Slave database* telah memiliki salinan data terkini melalui replikasi.
- Aplikasi hanya terhubung ke ProxySQL, sehingga tidak perlu mengetahui perubahan peran server.
- Pemulihan dapat dilakukan dengan cepat dan *downtime* dapat diminimalkan.

Hasil:
Operasional restoran dapat tetap berjalan normal dengan gangguan layanan yang sangat singkat.

2. Kegagalan *Slave Database*

Skenario:
Slave database mengalami gangguan sementara *master database* tetap berfungsi.

Yang Terjadi:

- ProxySQL mendeteksi bahwa *slave* tidak merespons.
- *Query* baca tidak lagi dikirim ke *slave*.
- Seluruh *query* (baca dan tulis) sementara ditangani oleh *master database*.

Alasan Mekanisme Ini Efektif:

- *Master database* dapat beroperasi secara mandiri tanpa bergantung pada *slave*.
- Tidak terjadi kehilangan data karena seluruh operasi tulis selalu dilakukan di *master*.
- *Slave* dapat diperbaiki dan disinkronkan ulang tanpa menghentikan sistem.

Hasil:
Tidak terjadi *downtime*, hanya penurunan performa kecil saat beban tinggi.

3. Gangguan Jaringan antara *Master* dan *Slave*

Skenario:
Koneksi jaringan antara *master* dan *slave database* terputus.

Yang Terjadi:

- *Slave database* tidak menerima pembaruan replikasi.
- ProxySQL hanya mengarahkan lalu lintas ke *master database*.
- *Slave* tetap dalam mode *read-only* dan tidak diizinkan menerima operasi tulis.

Alasan Mekanisme Ini Efektif:

- Hanya satu database yang diizinkan menangani operasi tulis dalam satu waktu.
- Hal ini mencegah konflik data dan kondisi *split-brain*.
- Setelah jaringan pulih, replikasi akan berjalan kembali secara otomatis.

Hasil:
Sistem tetap beroperasi dan konsistensi data tetap terjaga.

4. Kegagalan Sistem Total (*Disaster Scenario*)

Skenario:
Baik *master* maupun *slave database* tidak dapat diakses, misalnya akibat pemadaman listrik total atau kerusakan fisik.

Yang Terjadi:

- Database dipulihkan dari *backup* terbaru.
- Jika diperlukan, *binary log* diterapkan untuk memulihkan transaksi terakhir.
- Sistem diaktifkan kembali setelah proses pemulihan selesai.

Alasan Mekanisme Ini Efektif:

- *Backup* disimpan terpisah dari server database utama.
- *Backup* rutin mengurangi potensi kehilangan data.
- Prosedur pemulihan manual memastikan data tetap dapat dikembalikan dalam kondisi ekstrem.

Hasil:
Sistem dapat dipulihkan, meskipun dengan waktu *downtime* yang lebih lama dibandingkan *failover* normal.

Strategi Backup dan Recovery

1. Gambaran Umum Strategi *Backup*

Sistem DineFlow menggunakan strategi *backup* untuk melindungi data dari penghapusan tidak sengaja, kegagalan sistem, maupun kondisi bencana. Data penting restoran seperti pesanan, pembayaran, dan data pelanggan harus tetap dapat dipulihkan agar operasional tidak terganggu.

Strategi ini mengombinasikan *full backup* secara berkala dan pencatatan transaksi berkelanjutan (*binary log*) untuk menjaga keamanan data tanpa membebani performa sistem.

2. Full Database Backup

Deskripsi:
Full backup seluruh database dilakukan satu kali setiap hari di luar jam operasional.

Cara Kerja:

- Seluruh tabel, data, dan struktur database dicadangkan.
- *Backup* disimpan di penyimpanan lokal untuk pemulihan cepat.

- Salinan *backup* juga disimpan di *cloud storage* sebagai perlindungan tambahan.

Alasan Digunakan:

- *Full backup* mudah dilakukan dan andal.
- Proses *backup* di malam hari tidak mengganggu jam sibuk restoran.
- Penyimpanan *offsite* melindungi data jika server *on-premise* rusak.

Tujuan:
Memungkinkan pemulihan database secara penuh saat terjadi kegagalan sistem.

3. Binary Log Backup (Incremental Backup)

Deskripsi:
Binary log mencatat setiap perubahan data yang terjadi di database.

Cara Kerja:

- Semua operasi *insert*, *update*, dan *delete* dicatat dalam *binary log*.
- *Log* ini dibackup secara berkala dan disimpan terpisah.
- Saat pemulihan, *binary log* dijalankan kembali setelah *full backup* dipulihkan.

Alasan Digunakan:

- Mengurangi potensi kehilangan data di antara *full backup*.
- Mendukung *point-in-time recovery*.
- Lebih efisien dibanding melakukan *full backup* terlalu sering.

Tujuan:
Memungkinkan pemulihan data hingga waktu tertentu sebelum kesalahan terjadi.

4. Weekly Physical Backup

Deskripsi:
Physical backup file database dilakukan setiap minggu menggunakan metode pencadangan cepat.

Cara Kerja:

- File database disalin langsung dari level penyimpanan.
- *Backup* dikompresi dan disimpan secara aman.
- *Backup* lama dihapus sesuai kebijakan retensi.

Alasan Digunakan:

- *Physical backup* lebih cepat dipulihkan dibanding *logical backup*.

- Menyediakan opsi pemulihan tambahan jika *logical backup* gagal.

Tujuan:
Mempercepat proses pemulihan pada kegagalan besar.

5. Skenario *Recovery*

5.1 Pemulihan Database Penuh

Skenario:
Database tidak dapat digunakan akibat kegagalan besar.

Langkah Pemulihan:

1. Memulihkan *full backup* terbaru.
2. Menerapkan *binary log* untuk memulihkan transaksi terakhir.
3. Memverifikasi integritas data sebelum sistem digunakan kembali.

Alasan:
Metode ini mengembalikan database sedekat mungkin dengan kondisi terakhir yang normal.

5.2 *Point-in-Time Recovery*

Skenario:
Data terhapus atau berubah secara tidak sengaja pada waktu tertentu.

Langkah Pemulihan:

1. Memulihkan *full backup* terbaru.
2. Menjalankan *binary log* hingga waktu sebelum kesalahan terjadi.
3. Menghentikan proses pemulihan pada titik waktu yang diinginkan.

Alasan:
Mencegah data yang salah atau rusak ikut dipulihkan kembali.

5.3 Pemulihan Satu Tabel

Skenario:
Hanya satu tabel yang rusak atau terhapus.

Langkah Pemulihan:

1. Memulihkan tabel terkait ke database sementara.
2. Menyalin tabel yang benar ke database produksi.
3. Memastikan data telah pulih dengan benar.

Alasan:
Lebih efisien dibanding memulihkan seluruh database untuk masalah kecil.

6. Pengujian dan Keandalan *Backup*

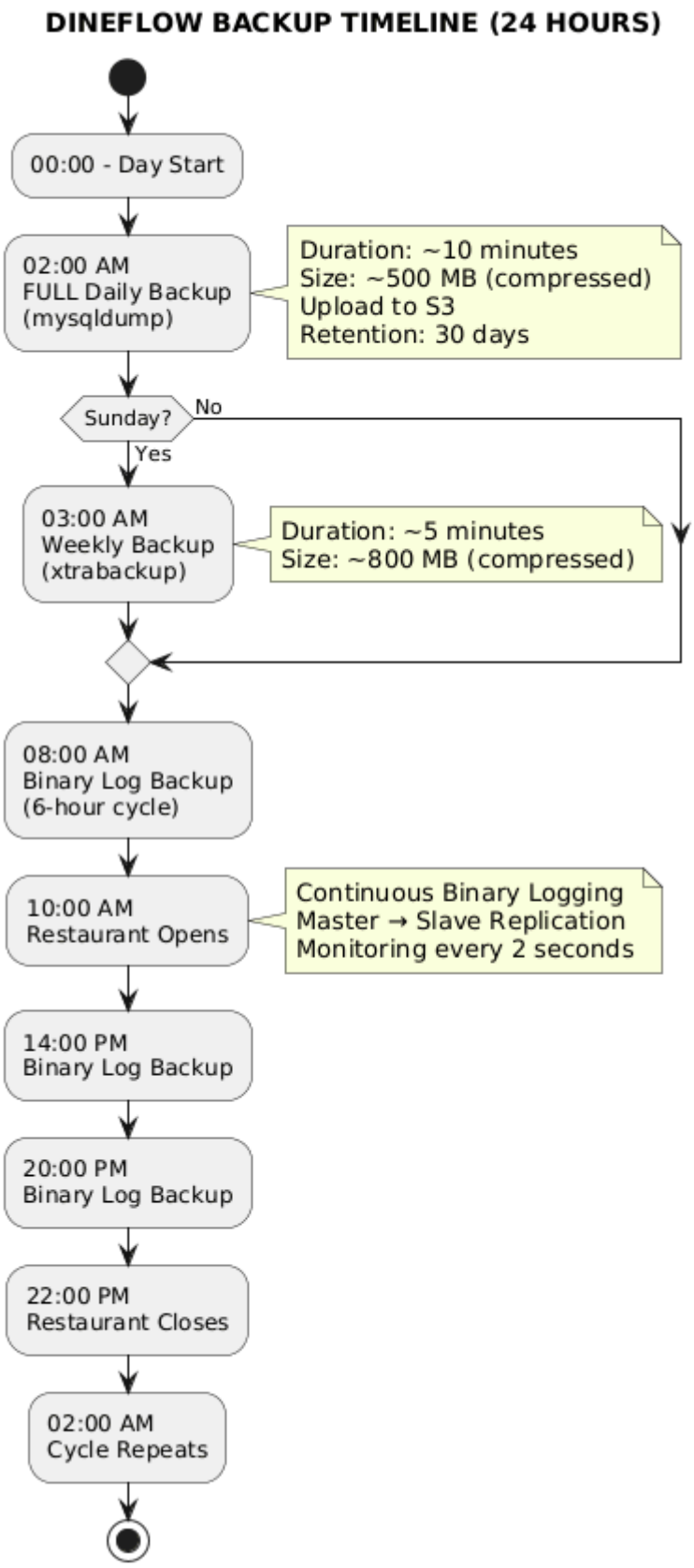
Backup diuji secara berkala untuk memastikan dapat dipulihkan dengan baik.

Pengujian Meliputi:

- Pemeriksaan integritas file *backup*.
- Pemulihan *backup* ke lingkungan pengujian.
- Validasi konsistensi data setelah pemulihan.

Alasan Pengujian Penting:
Backup yang tidak dapat dipulihkan tidak memiliki nilai. Pengujian rutin memastikan keandalan sistem.

Backup Timeline Flow (Daily Backup)



Penjelasan *Backup Timeline*

1. Timeline *Backup* Harian

Waktu: Dini hari (di luar jam operasional)

Yang Terjadi:

- *Full backup* database dibuat.
- Seluruh tabel, data, dan struktur database dicadangkan.
- File *backup* disimpan di penyimpanan lokal dan diunggah ke *cloud storage*.

Alasan:

- *Full backup* harian menyediakan titik pemulihan yang stabil.
- Proses di luar jam operasional mencegah penurunan performa saat transaksi berlangsung.

4. *Backup* Berkelanjutan Selama Jam Operasional

Waktu: Sepanjang jam buka restoran

Yang Terjadi:

- Setiap perubahan data dicatat dalam *MySQL binary log*.
- *Binary log* disalin dan disimpan secara berkala.
- Seluruh transaksi seperti pesanan, pembayaran, dan pembaruan data terekam.

Alasan:

- Menjamin transaksi terbaru tidak hilang.
- Mendukung pemulihan hingga waktu tertentu (*point-in-time recovery*).
- Mengurangi kebutuhan *full backup* yang terlalu sering.

3. *Binary Log Backup* Berkala

Waktu: Setiap beberapa jam

Yang Terjadi:

- File *binary log* disalin dari server database.
- *Log* lama diarsipkan atau dihapus sesuai kebijakan retensi.
- *Log* disimpan terpisah dari database utama.

Alasan:

- Mencegah penggunaan ruang disk berlebih.
- Memastikan *binary log* selalu tersedia untuk proses pemulihan.

4. Timeline *Backup* Mingguan

Waktu: Satu kali per minggu saat aktivitas rendah

Yang Terjadi:

- *Physical backup* database dibuat.
- *Backup* dikompresi dan disimpan secara aman.
- *Backup* mingguan lama dihapus sesuai kebijakan retensi.

Alasan:

- *Physical backup* memungkinkan pemulihan lebih cepat pada kegagalan besar.
- Memberikan lapisan perlindungan tambahan selain *backup* harian.

5. Retensi dan Pembersihan *Backup*

Yang Terjadi:

- *Backup* harian disimpan untuk jangka waktu terbatas.
- *Backup* mingguan disimpan lebih lama.
- *Backup* lama dihapus secara otomatis.

Alasan:

- Menghindari penggunaan penyimpanan yang berlebihan.
- Menjaga sistem *backup* tetap rapi dan mudah dikelola.