TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**NGUYỄN TRẦN HOÀNG NHÂN – 523H0164**

**BÀI TẬP LỚN HỆ CƠ SỞ DỮ LIỆU  
2024 - 2025**

**HỆ CƠ SỞ DỮ LIỆU**

**THÀNH PHỐ HỒ CHÍ MINH, 2024**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**NGUYỄN TRẦN HOÀNG NHÂN – 523H0164**

**BÀI TẬP LỚN CƠ SỞ DỮ LIỆU**

**HỆ CƠ SỞ DỮ LIỆU**

Người hướng dẫn

**Thầy Dương Hớn Minh**

**THÀNH PHỐ HỒ CHÍ MINH, 2024**

**Mục Lục**

[CHAPTER 1. PHẦN 1 1](#_Toc21791)

[1.1 Đặc tả nghiệp vụ cho quản lý và cho thuê tổ chức sự kiện: 1](#_Toc51)

[1.2 Mô hình ERD 2](#_Toc7035)

[1.3 Mô hình quan hệ 3](#_Toc28485)

[1.4 Các mối quan hệ trong mô hình 3](#_Toc20854)

[1.4.1 Quan hệ 1 - n: 3](#_Toc26345)

[1.4.2 Quan hệ n - n: 4](#_Toc13417)

[1.4.3 Quan hệ cha con: 4](#_Toc7312)

[1.4.4 Quan hệ 1 - 1: 4](#_Toc17305)

[1.4.5 Quan hệ thực thể mạnh và thực thể yếu: 4](#_Toc31666)

[1.5 Sử dụng câu lệnh SQL để thực hiện: 5](#_Toc10430)

[1.5.1 Tạo các bảng: 5](#_Toc19136)

[1.5.2 Tạo ít nhất 2 function/procedure để thêm dữ liệu vào các bảng: 11](#_Toc11702)

[1.5.3 Tạo ít nhất 1 trigger để kiểm tra ràng buộc khóa ngoại và ràng buộc miền giá trị: 18](#_Toc7310)

[CHAPTER 2. PHẦN 2 25](#_Toc28875)

[2.1 Chuyển đổi mô hình ERD sang mô hình quan hệ 25](#_Toc6648)

[2.1.1 Tổng quan: 25](#_Toc21558)

[2.1.2 Giải thích từng thành phần chính 25](#_Toc19928)

[2.1.3 Tóm tắt các bước chương trình thực hiện: 51](#_Toc19151)

[2.1.4 Hình ảnh các testcase: 53](#_Toc11655)

[2.2 Tìm khóa của lược đồ cơ sở dữ liệu: 54](#_Toc29435)

[2.2.1 Tổng quan: 54](#_Toc10873)

[2.2.2 Giải thích các thành phần chính: 55](#_Toc14944)

[2.2.3 Tóm tắt các bước chương trình thực hiện: 66](#_Toc15803)

[2.2.4 Hình ảnh các testcase: 68](#_Toc22502)

**Danh Sách Hình Ảnh**

[Hình 1: Mô hình ERD quản lý cho thuê tổ chức sự kiện 3](#_Toc13315)

[Hình 2: Mô hình quan hệ quản lý cho thuê tổ chức sự kiện 3](#_Toc17313)

[Hình 3: Bảng khách hàng 5](#_Toc6344)

[Hình 4: Bảng Hợp Đồng 5](#_Toc26875)

[Hình 5: Bảng Sự Kiện 6](#_Toc13918)

[Hình 6: Bảng Dịch Vụ 7](#_Toc19825)

[Hình 7: Bảng Chi Tiết Dịch Vụ Hợp Đồng 7](#_Toc7929)

[Hình 8: Bảng Loa 8](#_Toc26830)

[Hình 9: Bảng Đèn 8](#_Toc2038)

[Hình 10: Bảng Nhà Bánh Ú 9](#_Toc19638)

[Hình 11: Bảng Nhà Tiền Chế 9](#_Toc3903)

[Hình 12: Bảng Cổng Hội Chợ 10](#_Toc28404)

[Hình 13: Bảng Thanh Toán 10](#_Toc3287)

[Hình 14: Bảng Nhân Sự 10](#_Toc9053)

[Hình 15: Bảng Phân Công 11](#_Toc22955)

[Hình 16: Hàm thủ tục để thêm khách hàng 12](#_Toc10705)

[Hình 17: Hàm thủ tục để thêm hợp đồng 14](#_Toc32291)

[Hình 18: Hàm thủ tục để thêm sự kiện 16](#_Toc6417)

[Hình 19: Trigger kiểm tra khi thêm và cập nhập khách hàng 18](#_Toc3592)

[Hình 20: Trigger kiểm tra khi thêm và cập nhập hợp đồng 20](#_Toc30092)

[Hình 21: Trigger kiểm tra khi thêm và cập nhập sự kiện 22](#_Toc25780)

[Hình 22: Testcase input cho quan hệ 1 - n 54](#_Toc31570)

[Hình 23: Testcase output cho quan hệ 1 - n 54](#_Toc19369)

[Hình 24: Testcase input cho quan hệ n - n 54](#_Toc7363)

[Hình 25: Testcase output cho quan hệ n - n 54](#_Toc9987)

[Hình 26: Testcase input cho quan hệ 1 - 1 54](#_Toc967)

[Hình 27: Testcase output cho quan hệ 1 - 1 54](#_Toc24165)

[Hình 28: Testcase input cho thuộc tính đa trị 54](#_Toc4478)

[Hình 29: Testcase output cho thuộc tính đa trị 54](#_Toc11848)

[Hình 30: Testcase cho input quan hệ cha - con 54](#_Toc30933)

[Hình 31: Testcase cho output quan hệ cha - con 54](#_Toc11487)

[Hình 32: Test case input cho tìm khóa của lược đồ CSDL 68](#_Toc12898)

[Hình 33: Test case output cho tìm khóa của lược đồ CSDL 69](#_Toc26339)

# PHẦN 1

## Đặc tả nghiệp vụ cho quản lý và cho thuê tổ chức sự kiện:

Hệ thống quản lý cho thuê và tổ chức sự kiện được thiết kế để hỗ trợ các hoạt động từ việc quản lý khách hàng, ký kết hợp đồng, tổ chức sự kiện, cung cấp dịch vụ, đến thanh toán và quản lý nhân sự liên quan. Hệ thống giúp đảm bảo mọi thông tin được quản lý chặt chẽ, đồng bộ, và hỗ trợ các hoạt động ra quyết định một cách hiệu quả.

Quản lý khách hàng là chức năng đầu tiên của hệ thống, lưu trữ thông tin chi tiết về khách hàng bao gồm mã khách hàng (mã định danh duy nhất), tên, địa chỉ, số điện thoại và email. Đây là cơ sở để theo dõi những khách hàng sử dụng dịch vụ. Một khách hàng có thể ký nhiều hợp đồng với công ty, và tất cả thông tin liên quan sẽ được ghi nhận để quản lý lâu dài.

Quản lý hợp đồng để lưu trữ thông tin về các hợp đồng dịch vụ. Mỗi hợp đồng được định danh bằng mã hợp đồng, liên kết trực tiếp với mã khách hàng. Hợp đồng cũng bao gồm các thông tin như ngày ký kết, tổng giá trị và trạng thái hợp đồng. Đặc biệt, một hợp đồng có thể liên quan đến một sự kiện nhưng không bắt buộc. Thông qua chức năng này, công ty có thể theo dõi các hợp đồng đang thực hiện, các hợp đồng đã hoàn thành hoặc hủy bỏ.

Quản lý sự kiện cho phép công ty tổ chức các sự kiện theo hợp đồng đã ký. Mỗi sự kiện được định danh bằng mã sự kiện và bắt buộc phải liên kết với một hợp đồng cụ thể. Các thông tin liên quan đến sự kiện bao gồm tên sự kiện, ngày tổ chức, ngày kết thúc, địa điểm và mô tả chi tiết. Điều này đảm bảo rằng công ty có thể dễ dàng theo dõi từng sự kiện, từ giai đoạn chuẩn bị đến khi hoàn tất. Chức năng này đặc biệt quan trọng khi công ty phải phối hợp nhiều bộ phận để tổ chức sự kiện đúng tiến độ và chất lượng.

Hệ thống cũng cung cấp chức năng quản lý dịch vụ, theo dõi danh sách các dịch vụ được cung cấp như thiết bị âm thanh, ánh sáng, nhà bánh ú, nhà tiền chế, và cổng hội chợ. Mỗi dịch vụ được định danh bằng mã dịch vụ, có thông tin chi tiết như tên, đơn giá, mô tả, đơn vị tính và tình trạng hiện tại. Điều này giúp công ty tối ưu hóa việc quản lý tài nguyên và đảm bảo dịch vụ luôn sẵn sàng đáp ứng nhu cầu.

Khi một hợp đồng được ký, chi tiết các dịch vụ thuê sẽ được lưu trữ thông qua chức năng quản lý chi tiết dịch vụ hợp đồng. Mỗi mục chi tiết bao gồm mã hợp đồng, mã dịch vụ, số lượng, đơn giá và thành tiền (tính bằng số lượng nhân với đơn giá). Thông tin này không chỉ hỗ trợ tính toán tổng giá trị hợp đồng mà còn đảm bảo các tài nguyên được phân bổ chính xác và minh bạch.

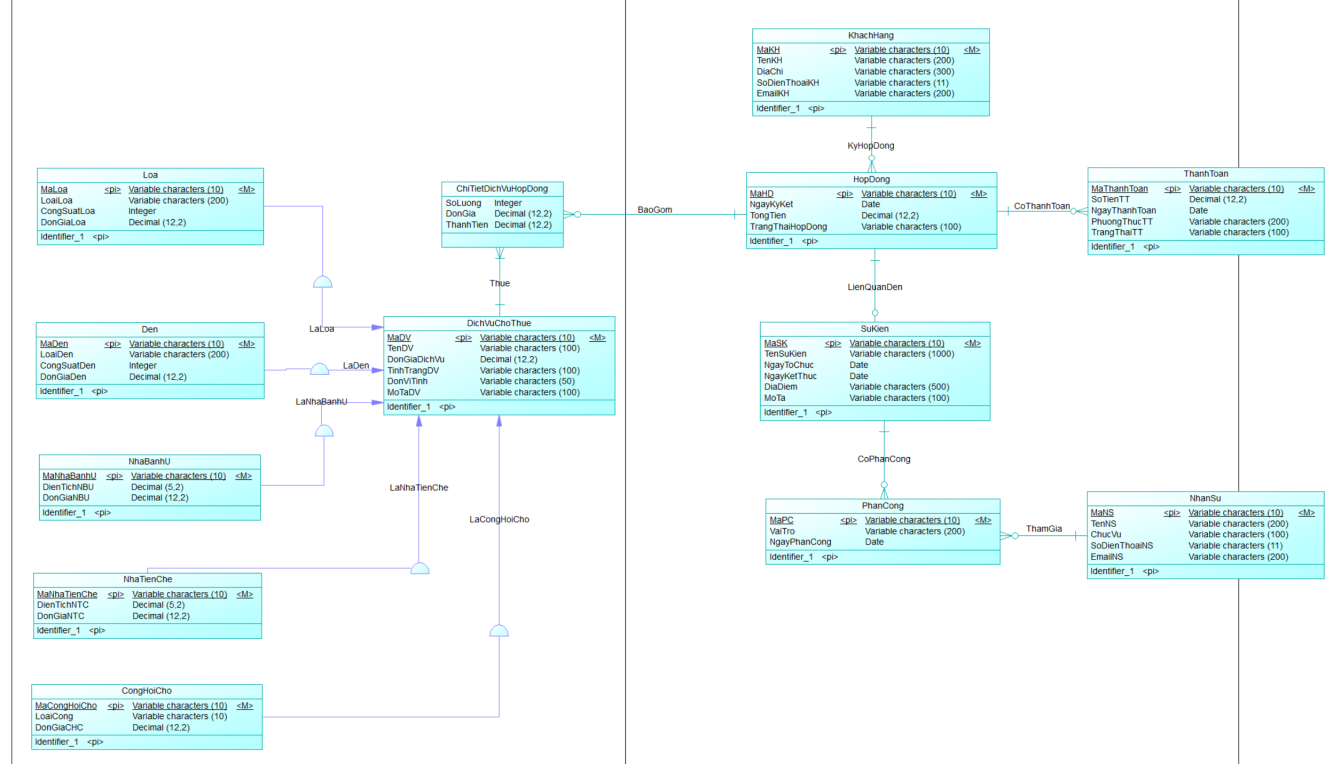
Để đảm bảo sự kiện được tổ chức hiệu quả, hệ thống hỗ trợ quản lý nhân sự và phân công nhiệm vụ. Mỗi nhân sự được định danh bằng mã nhân sự, cùng với thông tin về tên, chức vụ, số điện thoại và email. Khi một sự kiện diễn ra, nhân sự sẽ được phân công cụ thể với vai trò rõ ràng, ví dụ như quản lý sự kiện, kỹ thuật viên âm thanh, hoặc hỗ trợ ánh sáng. Hệ thống theo dõi các phân công này để đảm bảo nhân sự thực hiện đúng nhiệm vụ.

Chức năng quản lý thanh toán giúp theo dõi các khoản thanh toán liên quan đến hợp đồng. Mỗi giao dịch thanh toán được định danh bằng mã thanh toán và liên kết với mã hợp đồng. Hệ thống ghi nhận số tiền thanh toán, ngày thực hiện, phương thức thanh toán và trạng thái thanh toán. Điều này đảm bảo rằng các khoản thanh toán được thực hiện đầy đủ và đúng hạn, tránh nhầm lẫn hoặc thiếu sót.

Cuối cùng, hệ thống duy trì các mối quan hệ chặt chẽ giữa các thành phần. Khách hàng có thể ký nhiều hợp đồng, mỗi hợp đồng có thể liên quan đến một sự kiện và sử dụng nhiều dịch vụ. Dịch vụ có thể bao gồm các thiết bị cụ thể như loa, đèn, hoặc nhà bạt, được quản lý chi tiết để đáp ứng yêu cầu của từng sự kiện. Nhân sự được phân công cụ thể cho từng sự kiện để đảm bảo chất lượng tổ chức. Thanh toán được thực hiện và ghi nhận đầy đủ để hoàn thành hợp đồng.

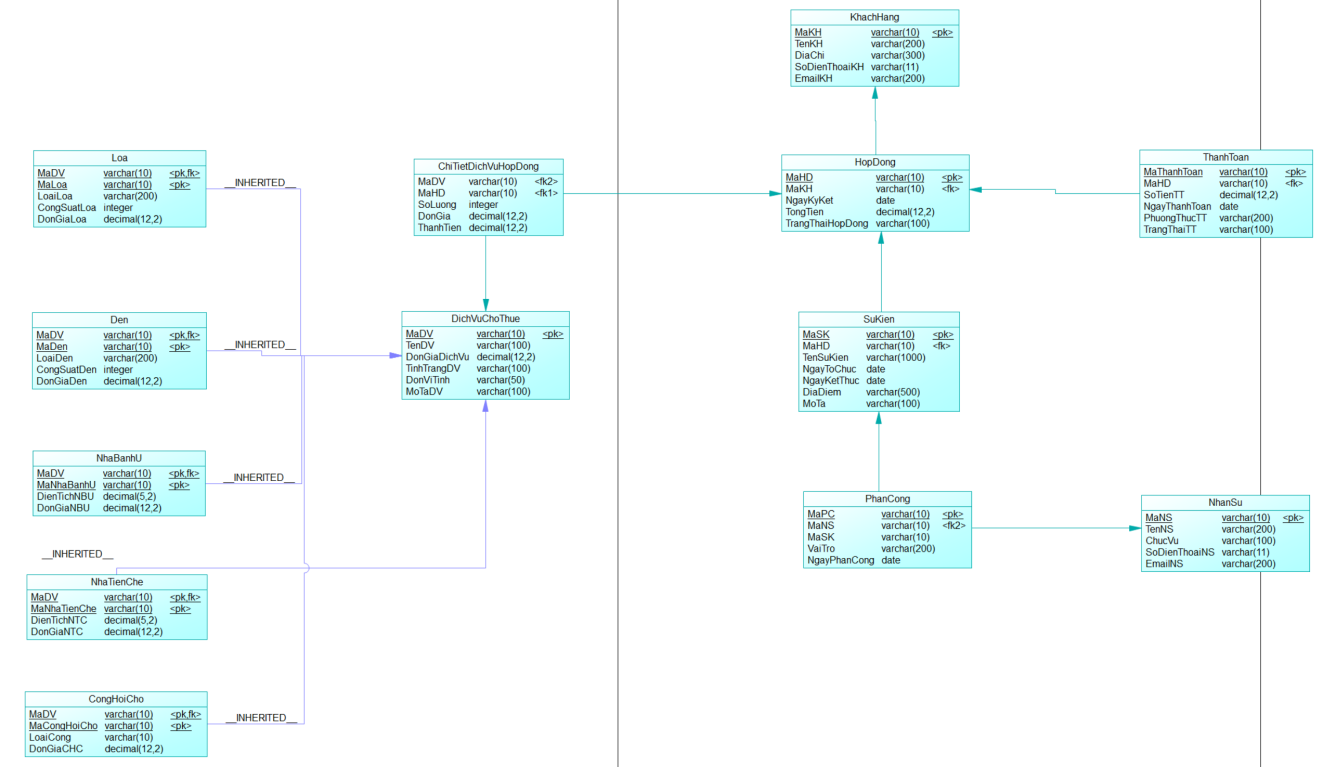
Hệ thống quản lý này không chỉ giúp công ty theo dõi toàn bộ quy trình từ ký hợp đồng, tổ chức sự kiện, đến thanh toán mà còn hỗ trợ lập báo cáo và phân tích dữ liệu. Điều này đảm bảo rằng công ty có thể đáp ứng tốt nhất nhu cầu của khách hàng, quản lý tài nguyên hiệu quả và nâng cao chất lượng dịch vụ.

## Mô hình ERD



Hình 1: Mô hình ERD quản lý cho thuê tổ chức sự kiện

## Mô hình quan hệ



Hình 2: Mô hình quan hệ quản lý cho thuê tổ chức sự kiện

## Các mối quan hệ trong mô hình

### Quan hệ 1 - n:

* Khách Hàng và Hợp Đồng: Mỗi khách hàng có thể ký nhiều hợp đồng, nhưng mỗi hợp đồng chỉ thuộc về một khách hàng.
* Hợp Đồng và Thanh Toán: Mỗi hợp đồng có thể có nhiều lần thanh toán, nhưng mỗi thanh toán chỉ thuộc về một hợp đồng.
* Hợp Đồng và Chi Tiết Dịch Vụ Hợp Đồng: Mỗi hợp đồng có thể bao gồm nhiều chi tiết dịch vụ, nhưng mỗi chi tiết dịch vụ thuộc về một hợp đồng.
* Nhân Sự và Phân Công: Mỗi nhân sự có thể tham gia nhiều phân công, nhưng mỗi phân công chỉ thuộc về một nhân sự.
* Sự Kiện và Phân Công: Mỗi sự kiện có thể có nhiều phân công, nhưng mỗi phân công chỉ thuộc về một sự kiện.

### Quan hệ n - n:

* Hợp Đồng và Dịch Vụ: Mối quan hệ nhiều-nhiều giữa hợp đồng và dịch vụ được thể hiện qua bảng Chi Tiết Dịch Vụ Hợp Đồng.
* Nhân Sự và Hợp Đồng (hoặc Sự Kiện): Mối quan hệ nhiều-nhiều giữa nhân sự và hợp đồng/sự kiện được thể hiện qua bảng Phân Công.

### Quan hệ cha con:

* Dịch Vụ và Loa: Loa là một loại dịch vụ cụ thể.
* Dịch Vụ và Đèn: Đèn là một loại dịch vụ cụ thể.
* Dịch Vụ và Nhà Bánh Ú: Nhà Bánh Ú là một loại dịch vụ cụ thể.
* Dịch Vụ và Nhà Tiền Chế: Nhà tiền chế là một loại dịch vụ cụ thể.
* Dịch Vụ và Cổng Hội Chợ: Cổng hội chợ là một loại dịch vụ cụ thể.

### Quan hệ 1 - 1:

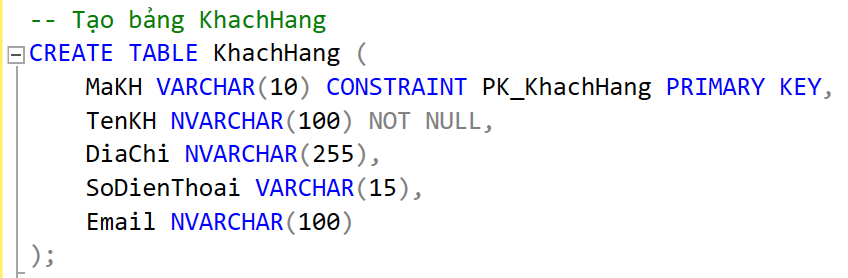
* Hợp Đồng và Sự Kiện: Mỗi hợp đồng có thể liên quan đến một sự kiện, và mỗi sự kiện chỉ thuộc về một hợp đồng.

### Quan hệ thực thể mạnh và thực thể yếu:

* Hợp Đồng và Chi Tiết Dịch Vụ Hợp Đồng: ChiTietDichVuHopDong là thực thể yếu, phụ thuộc vào HopDong (thực thể mạnh) để tồn tại.
* Dịch Vụ và Chi Tiết Dịch Vụ Hợp Đồng: ChiTietDichVuHopDong cũng phụ thuộc vào DichVu (thực thể mạnh).

## Sử dụng câu lệnh SQL để thực hiện:

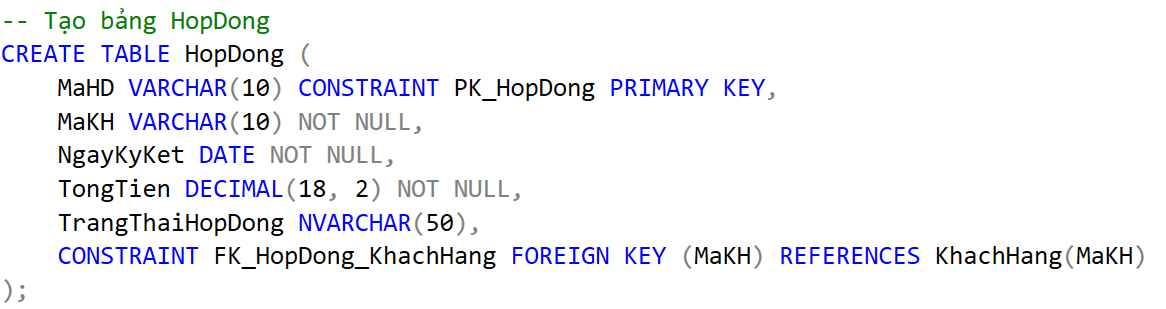
### Tạo các bảng:



Hình 3: Bảng khách hàng

* MaKH: Mã khách hàng, kiểu chuỗi tối đa 10 ký tự, là khóa chính của bảng.
* TenKH: Tên khách hàng, kiểu chuỗi Unicode tối đa 100 ký tự, không được để trống.
* DiaChi: Địa chỉ khách hàng, kiểu chuỗi Unicode tối đa 255 ký tự.
* SoDienThoai: Số điện thoại khách hàng, kiểu chuỗi tối đa 15 ký tự.
* Email: Email khách hàng, kiểu chuỗi Unicode tối đa 100 ký tự.

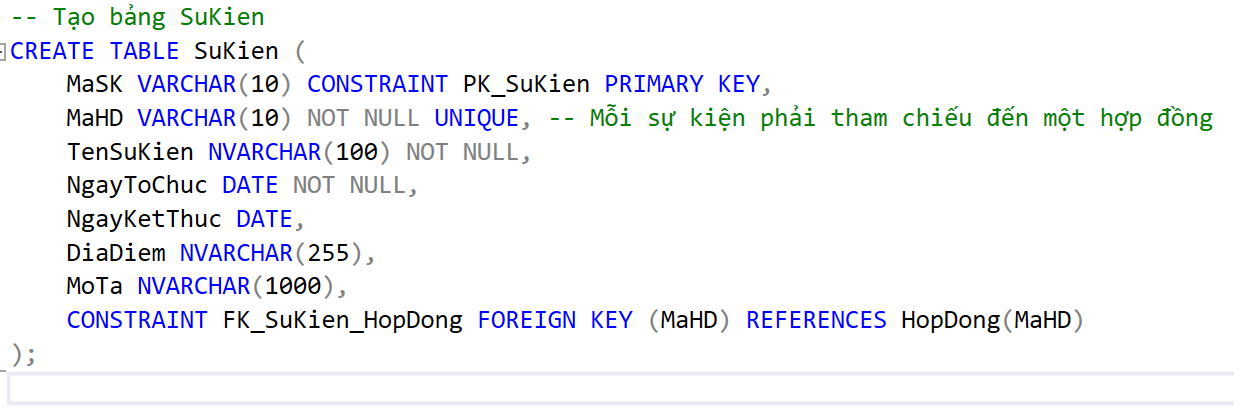
Chức năng: Lưu trữ thông tin của các khách hàng.



Hình 4: Bảng Hợp Đồng

* MaHD: Mã hợp đồng, kiểu chuỗi tối đa 10 ký tự, là khóa chính.
* MaKH: Mã khách hàng, không được để trống, khóa ngoại tham chiếu đến KhachHang(MaKH).
* NgayKyKet: Ngày ký kết hợp đồng, không được để trống.
* TongTien: Tổng tiền của hợp đồng, kiểu số thập phân, không được để trống.
* TrangThaiHopDong: Trạng thái của hợp đồng.
* Ràng buộc khóa ngoại: Đảm bảo rằng MaKH phải tồn tại trong bảng KhachHang.

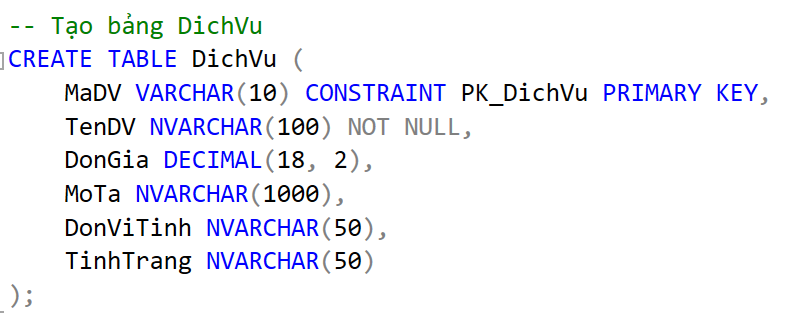
Chức năng: Lưu trữ thông tin các hợp đồng ký kết với khách hàng.



Hình 5: Bảng Sự Kiện

* MaSK: Mã sự kiện, là khóa chính.
* MaHD: Mã hợp đồng, không được để trống, duy nhất, khóa ngoại tham chiếu đến HopDong(MaHD).
* TenSuKien: Tên sự kiện, không được để trống.
* NgayToChuc: Ngày tổ chức sự kiện, không được để trống.
* NgayKetThuc: Ngày kết thúc sự kiện.
* DiaDiem: Địa điểm tổ chức.
* MoTa: Mô tả chi tiết về sự kiện.

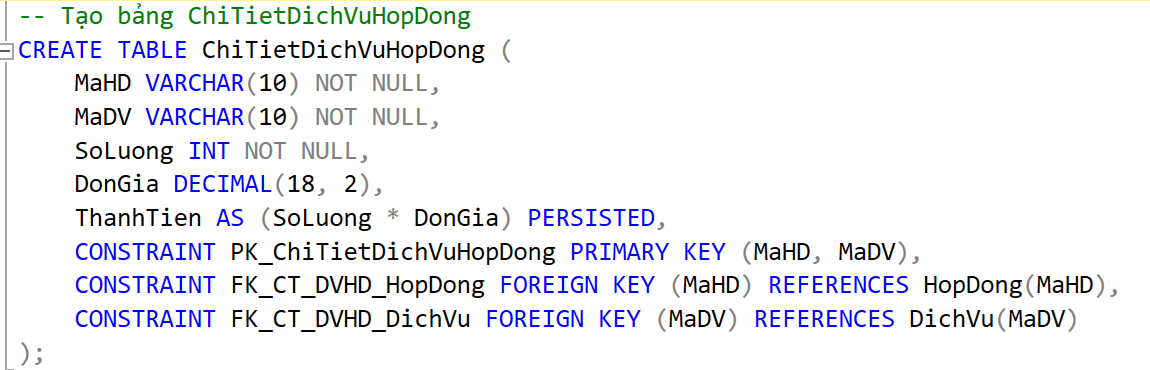
Chức năng: Lưu trữ thông tin về các sự kiện liên quan đến hợp đồng.



Hình 6: Bảng Dịch Vụ

* MaDV: Mã dịch vụ, là khóa chính.
* TenDV: Tên dịch vụ, không được để trống.
* DonGia: Đơn giá dịch vụ.
* MoTa: Mô tả dịch vụ.
* DonViTinh: Đơn vị tính (ví dụ: chiếc, bộ).
* TinhTrang: Tình trạng dịch vụ (ví dụ: còn hàng, hết hàng).

Chức năng: Lưu trữ thông tin về các dịch vụ cung cấp.



Hình 7: Bảng Chi Tiết Dịch Vụ Hợp Đồng

* MaHD: Mã hợp đồng, không được để trống, khóa ngoại.
* MaDV: Mã dịch vụ, không được để trống, khóa ngoại.
* SoLuong: Số lượng dịch vụ trong hợp đồng, không được để trống.
* DonGia: Đơn giá tại thời điểm ký hợp đồng.
* ThanhTien: Thành tiền, được tính bằng SoLuong \* DonGia, lưu trữ thực tế (PERSISTED).

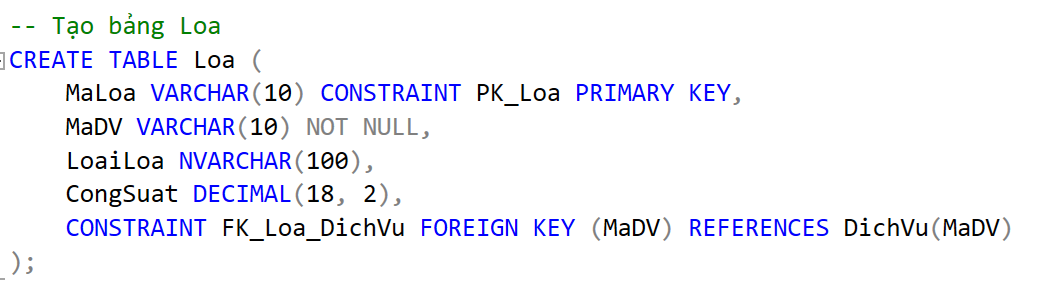
- PERSISTED dùng để:

+ Lưu trữ giá trị của cột tính toán được lưu vào cơ sở dữ liệu giống như các cột thông thường.

+ Tự động cập nhật nếu dữ liệu trong các cột liên quan thay đổi, giá trị của cột PERSISTED sẽ tự động được tính lại và cập nhật trong bảng.

* Khóa chính: Kết hợp MaHD và MaDV.

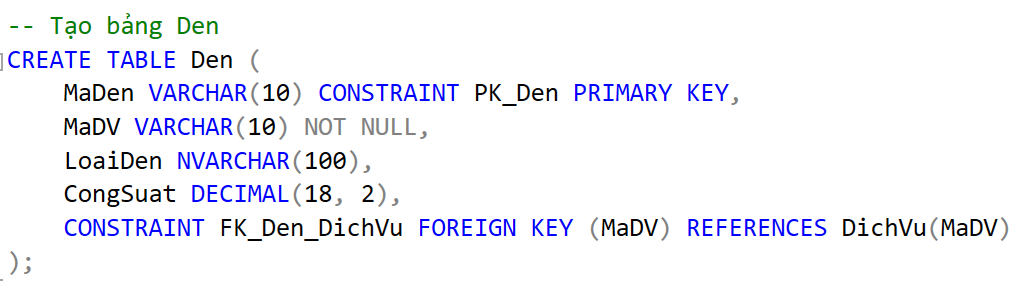
Chức năng: Lưu trữ chi tiết các dịch vụ trong từng hợp đồng, thể hiện mối quan hệ nhiều-nhiều giữa hợp đồng và dịch vụ.



Hình 8: Bảng Loa

* MaLoa: Mã loa, là khóa chính.
* MaDV: Mã dịch vụ, không được để trống, khóa ngoại.
* LoaiLoa: Loại loa (ví dụ: loa bass, loa treble).
* CongSuat: Công suất loa.

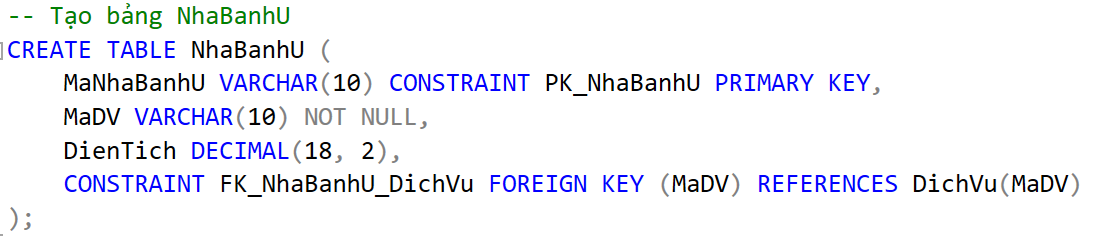
Chức năng: Lưu trữ thông tin chi tiết về dịch vụ loa, liên kết với bảng DichVu.



Hình 9: Bảng Đèn

* MaDen: Mã đèn, là khóa chính.
* MaDV: Mã dịch vụ, không được để trống, khóa ngoại.
* LoaiDen: Loại đèn (ví dụ: đèn LED, đèn halogen).
* CongSuat: Công suất đèn.

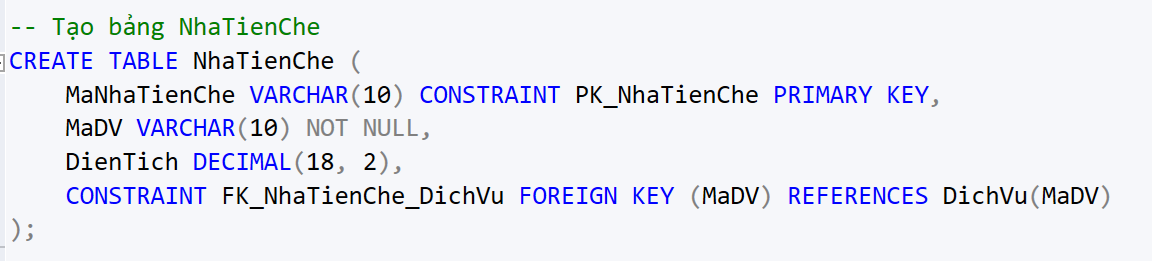
Chức năng: Lưu trữ thông tin chi tiết về dịch vụ đèn, liên kết với bảng DichVu.



Hình 10: Bảng Nhà Bánh Ú

* MaNhaBanhU: Mã nhà bạt chữ U, là khóa chính.
* MaDV: Mã dịch vụ, không được để trống, khóa ngoại.
* DienTich: Diện tích nhà bạt.

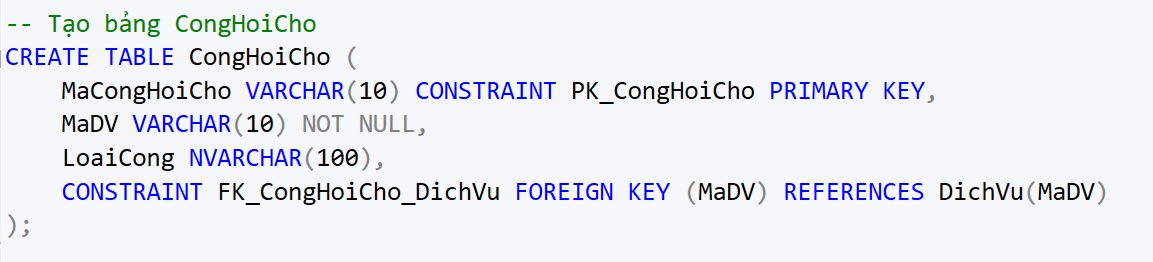
Chức năng: Lưu trữ thông tin về dịch vụ nhà bánh ú, liên kết với bảng DichVu.



Hình 11: Bảng Nhà Tiền Chế

* MaNhaTienChe: Mã nhà tiền chế, là khóa chính.
* MaDV: Mã dịch vụ, không được để trống, khóa ngoại.
* DienTich: Diện tích nhà tiền chế.

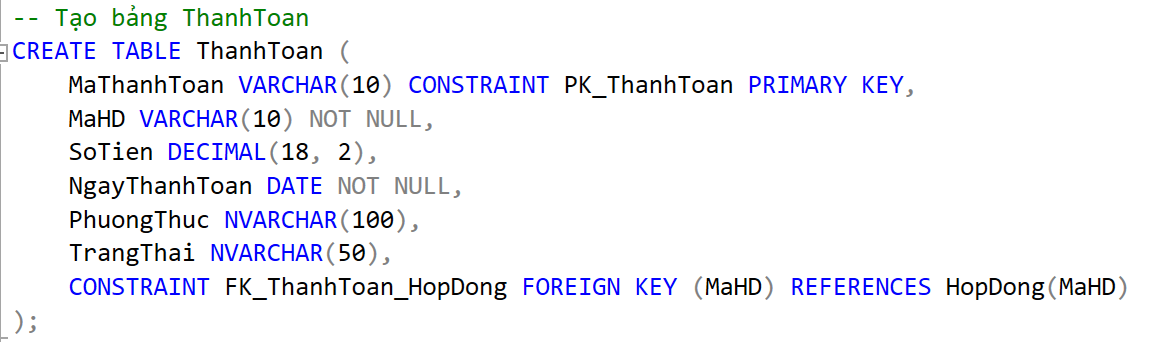
Chức năng: Lưu trữ thông tin về dịch vụ nhà tiền chế, liên kết với bảng DichVu.



Hình 12: Bảng Cổng Hội Chợ

* MaCongHoiCho: Mã cổng hội chợ, là khóa chính.
* MaDV: Mã dịch vụ, không được để trống, khóa ngoại.
* LoaiCong: Loại cổng (ví dụ: nhỏ, trung bình, lớn).

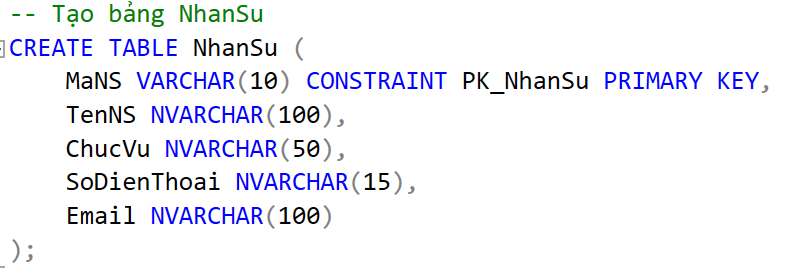
Chức năng: Lưu trữ thông tin về dịch vụ cổng hội chợ, liên kết với bảng DichVu.



Hình 13: Bảng Thanh Toán

* MaThanhToan: Mã thanh toán, là khóa chính.
* MaHD: Mã hợp đồng, không được để trống, khóa ngoại.
* SoTien: Số tiền thanh toán.
* NgayThanhToan: Ngày thanh toán, không được để trống.
* PhuongThuc: Phương thức thanh toán (ví dụ: tiền mặt, chuyển khoản).
* TrangThai: Trạng thái thanh toán (ví dụ: đã thanh toán, chưa thanh toán).

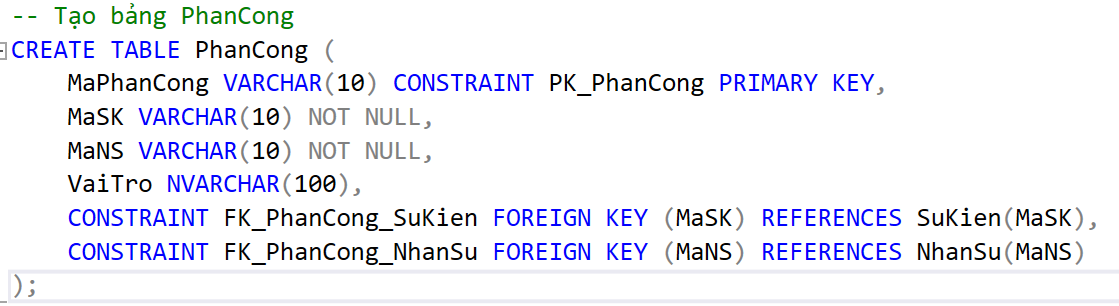
Chức năng: Lưu trữ thông tin các lần thanh toán cho mỗi hợp đồng.



Hình 14: Bảng Nhân Sự

* MaNS: Mã nhân sự, là khóa chính.
* TenNS: Tên nhân sự.
* ChucVu: Chức vụ (ví dụ: kỹ thuật viên, quản lý).
* SoDienThoai: Số điện thoại nhân sự.
* Email: Email nhân sự.

Chức năng: Lưu trữ thông tin về nhân viên trong công ty.

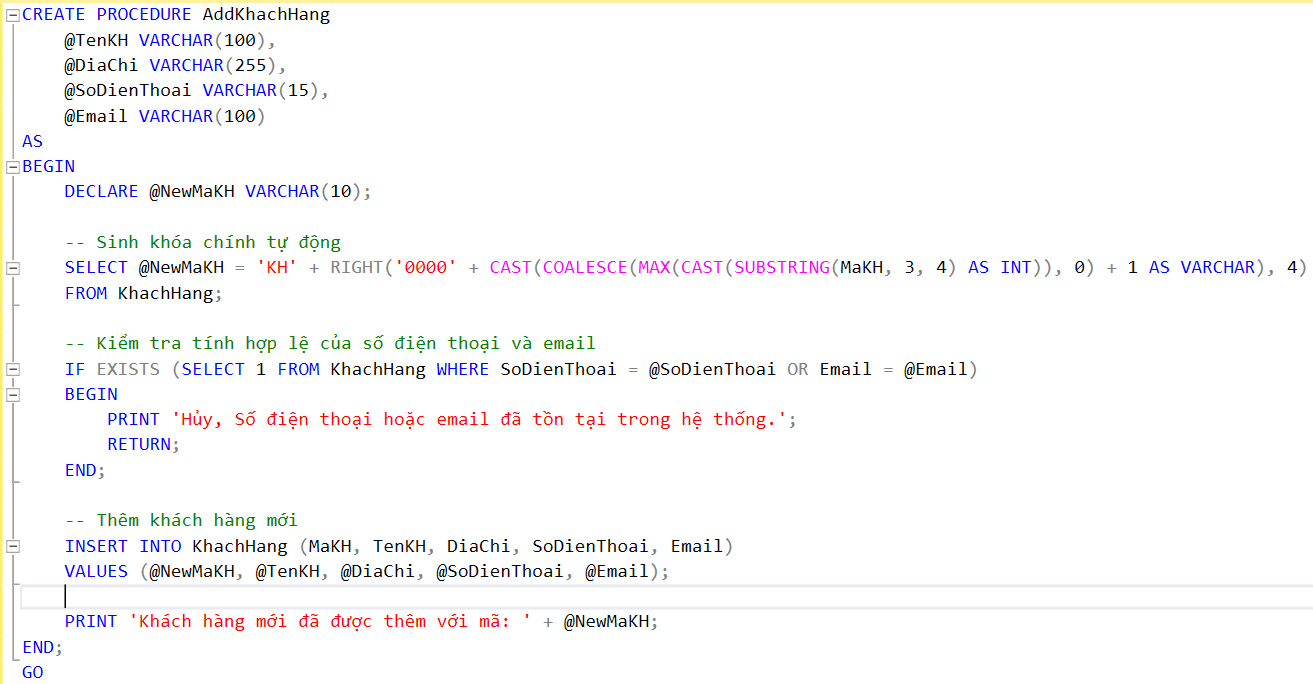


Hình 15: Bảng Phân Công

* MaPhanCong: Mã phân công, là khóa chính.
* MaSK: Mã sự kiện, không được để trống, khóa ngoại.
* MaNS: Mã nhân sự, không được để trống, khóa ngoại.
* VaiTro: Vai trò của nhân sự trong hợp đồng.

Chức năng: Lưu trữ thông tin về việc phân công nhân sự cho các sự kiện, thể hiện mối quan hệ giữa nhân sự và sự kiện.

### Tạo ít nhất 2 function/procedure để thêm dữ liệu vào các bảng:



Hình 16: Hàm thủ tục để thêm khách hàng

**Bước 1: Khai báo tham số đầu vào:**

@TenKH NVARCHAR(100),

@DiaChi NVARCHAR(255),

@SoDienThoai VARCHAR(15),

@Email VARCHAR(100)

- Các tham số nhận thông tin khách hàng cần thêm.

- Những tham số này nhận giá trị từ người gọi Procedure.

**Bước 2: Sinh mã khách hàng tự động:**

**Ta có các bước như sau:**

1. Trích xuất phần số từ mã khách hàng (MaKH).
2. Lấy số thứ tự lớn nhất hiện có, hoặc 0 nếu bảng rỗng.
3. Tăng thêm 1 để tạo số thứ tự mới.
4. Định dạng số thứ tự thành chuỗi có 4 chữ số, thêm tiền tố 'KH' để tạo mã hoàn chỉnh.

DECLARE @NewMaKH VARCHAR(10);

SELECT @NewMaKH = 'KH' + RIGHT('0000' + CAST(COALESCE(MAX(CAST(SUBSTRING(MaKH, 3, 4) AS INT)), 0) + 1 AS VARCHAR), 4)

FROM KhachHang;

* Biến @NewMaKH dùng để lưu mã khách hàng mới được tạo tự động.

SUBSTRING(MaKH, 3, 4):

* Lấy 4 ký tự từ vị trí thứ 3 trong mã khách hàng (MaKH).
* Ví dụ: 'KH0001' → '0001'.

CAST(... AS INT):

* Chuyển chuỗi '0001' thành số nguyên 1 để thực hiện các phép tính.

MAX(...):

* Lấy giá trị số lớn nhất trong bảng KhachHang.
* Nếu bảng chứa 'KH0001', 'KH0002', thì kết quả là 2.

COALESCE(MAX(...), 0):

* Đảm bảo trả về 0 nếu bảng KhachHang trống.

+ 1:

* Tăng giá trị lớn nhất thêm 1 để tạo số thứ tự cho mã mới.
* Ví dụ: 2 + 1 = 3.

'0000' + ... và RIGHT(..., 4):

* Thêm chuỗi '0000' vào trước số thứ tự, rồi lấy 4 ký tự cuối.
* Ví dụ: 00003 → '0003'.

'KH' + ...:

* Thêm tiền tố 'KH' để tạo mã khách hàng hoàn chỉnh.
* Ví dụ: 'KH' + '0003' → 'KH0003'.

@NewMaKH =:

* Lưu mã khách hàng mới ('KH0003') vào biến @NewMaKH.

**Bước 3: Kiểm tra tính hợp lệ:**

IF EXISTS (SELECT 1 FROM KhachHang WHERE SoDienThoai = @SoDienThoai OR Email = @Email)

BEGIN

PRINT 'Hủy, Số điện thoại hoặc email đã tồn tại trong hệ thống.';

RETURN;

END;

- Kiểm tra xem số điện thoại hoặc email đã tồn tại trong bảng KhachHang hay chưa.

- Nếu tồn tại:

+ PRINT: Hiển thị thông báo lỗi.

+ RETURN: Dừng Procedure, không thêm dữ liệu.

**Bước 4: Thêm dữ liệu vào bảng:**

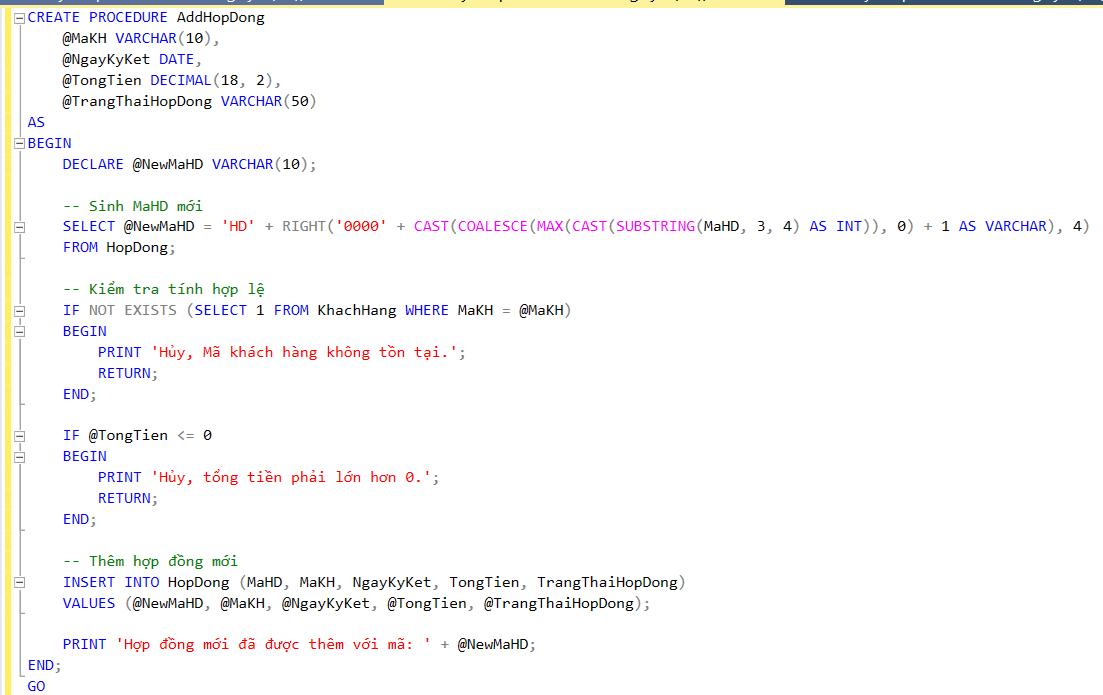
INSERT INTO KhachHang (MaKH, TenKH, DiaChi, SoDienThoai, Email)

VALUES (@NewMaKH, @TenKH, @DiaChi, @SoDienThoai, @Email);

PRINT 'Khách hàng mới đã được thêm với mã: ' + @NewMaKH;

- Chỉ thực hiện lệnh INSERT nếu kiểm tra trên không phát hiện lỗi.

- Hiển thị thông báo khi thêm khách hàng thành công.



Hình 17: Hàm thủ tục để thêm hợp đồng

**Bước 1: Khai báo tham số đầu vào:** @MaKH VARCHAR(10),

@NgayKyKet DATE,

@TongTien DECIMAL(18, 2),

@TrangThaiHopDong VARCHAR(50)

- Các tham số nhận thông tin sự kiện cần thêm.

**Bước 2: Sinh mã khách hàng tự động:**

DECLARE @NewMaHD VARCHAR(10);

SELECT @NewMaHD= 'HD'+ RIGHT ('0000'+

CAST(COALESCE(MAX(CAST(SUBSTRING(MaHD, 3, 4) AS INT)), 0) + 1 AS VARCHAR), 4)

FROM HopDong;

- Tương tự cách sinh mã khách hàng, mã hợp đồng có tiền tố HD.

**Bước 3: Kiểm tra tính hợp lệ:**

IF NOT EXISTS (SELECT 1 FROM KhachHang WHERE MaKH = @MaKH)

BEGIN

PRINT 'Hủy, Mã khách hàng không tồn tại.';

RETURN;

END;

IF @TongTien <= 0

BEGIN

PRINT 'Hủy, tổng tiền phải lớn hơn 0.';

RETURN;

END;

- Kiểm tra sự tồn tại của khách hàng trong bảng KhachHang.

- Đảm bảo tổng tiền phải lớn hơn 0.

**Bước 4: Thêm dữ liệu vào bảng:**

INSERT INTO HopDong (MaHD, MaKH, NgayKyKet, TongTien,

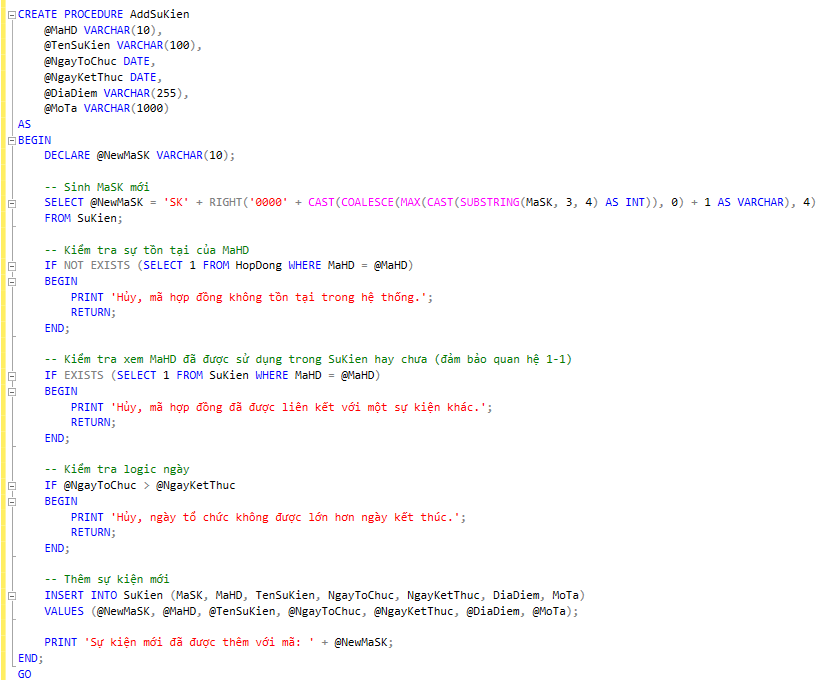
TrangThaiHopDong)

VALUES (@NewMaHD, @MaKH, @NgayKyKet, @TongTien, @TrangThaiHopDong);

PRINT 'Hợp đồng mới đã được thêm với mã: ' + @NewMaHD;

- Lệnh INSERT chỉ được thực thi khi tất cả các kiểm tra đều hợp lệ.

- In mã hợp đồng mới nếu thêm thành công.



Hình 18: Hàm thủ tục để thêm sự kiện

**Bước 1: Khai báo tham số đầu vào:**

@MaHD VARCHAR(10),

@TenSuKien VARCHAR(100),

@NgayToChuc DATE,

@NgayKetThuc DATE,

@DiaDiem VARCHAR(255),

@MoTa VARCHAR(1000)

- Các tham số nhận thông tin sự kiện cần thêm.

**Bước 2: Sinh mã khách hàng tự động:**

DECLARE @NewMaSK VARCHAR(10);

SELECT @NewMaSK = 'SK' + RIGHT('0000' + CAST(COALESCE(MAX(CAST(SUBSTRING(MaSK, 3, 4) AS INT)), 0) + 1 AS VARCHAR), 4)

FROM SuKien;

- Tương tự cách sinh mã khách hàng và hợp đồng, mã sự kiện có tiền tố SK.

**Bước 3: Kiểm tra tính hợp lệ:**

IF EXISTS (SELECT 1 FROM SuKien WHERE MaHD = @MaHD)

BEGIN

PRINT 'Hủy, mã hợp đồng đã được liên kết với một sự kiện khác.';

RETURN;

END;

IF @NgayToChuc > @NgayKetThuc

BEGIN  
 PRINT 'Hủy, ngày tổ chức không được lớn hơn ngày kết thúc.';

RETURN;

END;

- Đảm bảo hợp đồng liên kết tồn tại.

- Kiểm tra NgayToChuc phải nhỏ hơn hoặc bằng NgayKetThuc.

**Bước 4: Thêm dữ liệu vào bảng:**

INSERT INTO SuKien (MaSK, MaHD, TenSuKien, NgayToChuc, NgayKetThuc, DiaDiem, MoTa)

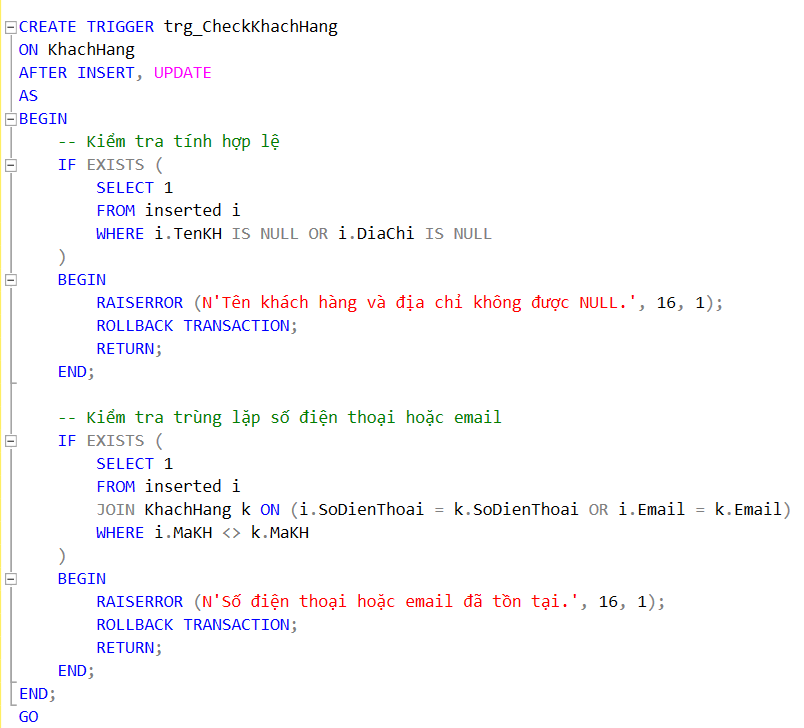
VALUES (@NewMaSK, @MaHD, @TenSuKien, @NgayToChuc, @NgayKetThuc, @DiaDiem, @MoTa);

PRINT 'Sự kiện mới đã được thêm với mã: ' + @NewMaSK;

- Lệnh INSERT chỉ được thực thi khi tất cả các kiểm tra đều hợp lệ.

- In mã hợp đồng mới nếu thêm thành công.

### Tạo ít nhất 1 trigger để kiểm tra ràng buộc khóa ngoại và ràng buộc miền giá trị:



Hình 19: Trigger kiểm tra khi thêm và cập nhập khách hàng

**Mục đích:**

- Trigger trg\_CheckKhachHang được thiết kế để thực hiện kiểm tra dữ liệu khi bảng KhachHang có sự thêm vào và cập nhập

**Bước 1: Kiểm tra tính hợp lệ của dữ liệu có null hay không:**

IF EXISTS (

SELECT 1

FROM inserted i

      WHERE i.TenKH IS NULL OR i.DiaChi IS NULL

)

BEGIN

        RAISERROR (N'Tên khách hàng và địa chỉ không được NULL.', 16, 1);

        ROLLBACK TRANSACTION;

        RETURN;

    END;

- Bảng inserted là bảng tạm chứa các bản ghi mới được chèn hoặc cập nhật.

- Trigger kiểm tra nếu bất kỳ giá trị nào trong các cột TenKH hoặc DiaChi là NULL.

- Nếu điều kiện đúng thì hủy bỏ giao dịch ROLLBACK TRANSACTION.

- Kết thúc trigger bằng RETURN.

**Bước 2: Kiểm tra trùng lặp số điện thoại hoặc email:**

IF EXISTS (

SELECT 1

FROM inserted I

JOIN KhachHang k ON (i.SoDienThoai = k.SoDienThoai OR i.Email = k.Email)

WHERE i.MaKH <> k.MaKH

)

BEGIN

RAISERROR (N'Số điện thoại hoặc email đã tồn tại.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END;

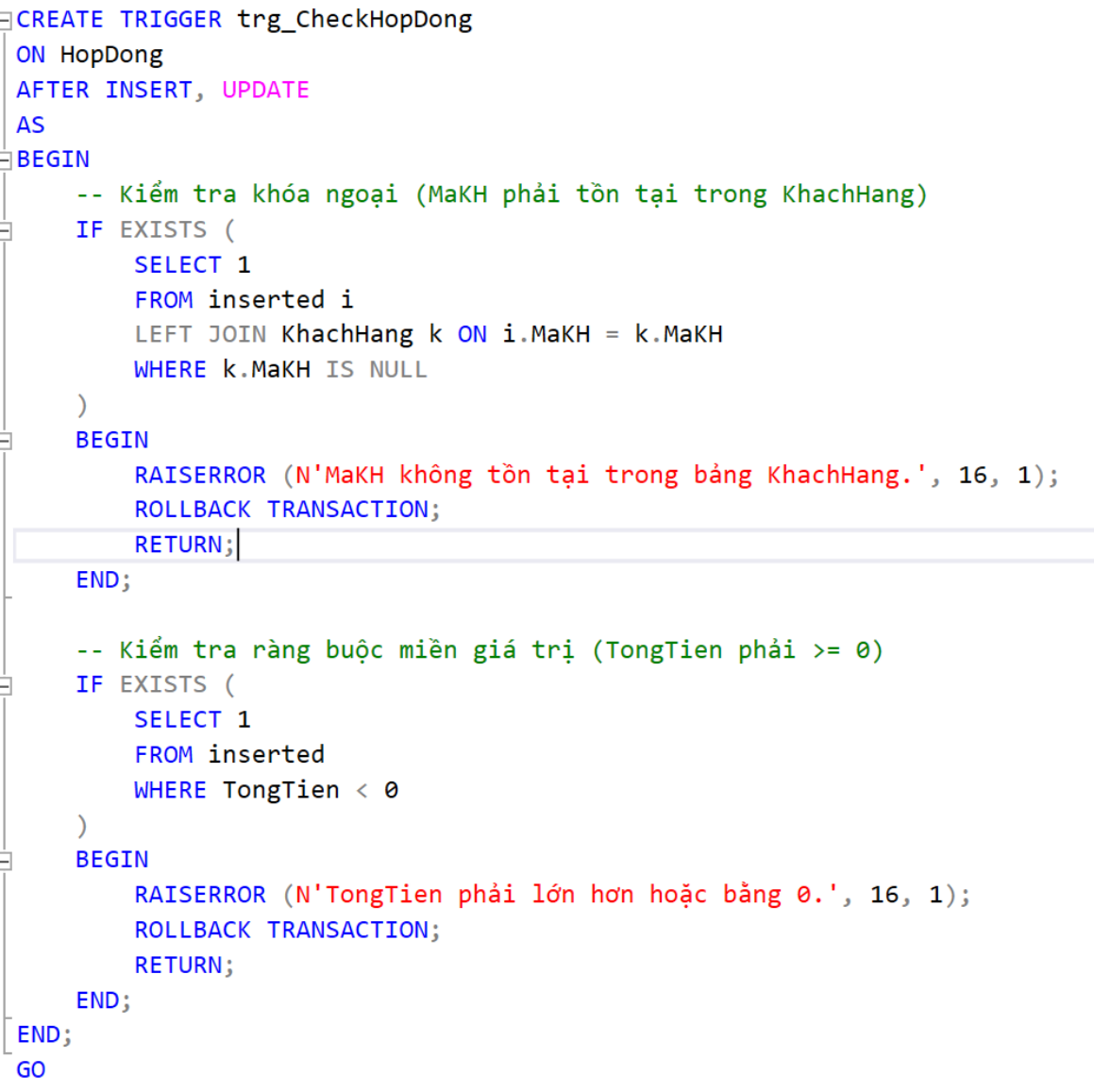
- Lệnh JOIN so sánh các bản ghi trong bảng inserted với bảng gốc KhachHang.

- Điều kiện i.MaKH <> k.MaKH đảm bảo loại trừ kiểm tra bản ghi hiện tại khi cập nhật.

- Nếu trùng lặp thì gửi thông báo lỗi qua RAISERROR.

- Hủy giao dịch ngay lập tức với lệnh ROLLBACK TRANSACTION.

- Dừng trigger với lệnh RETURN.



Hình 20: Trigger kiểm tra khi thêm và cập nhập hợp đồng

**Mục đích:**

- Trigger trg\_CheckHopDong được thiết kế để kiểm tra và đảm bảo tính toàn vẹn dữ liệu khi có các thao tác thêm mới hoặc cập nhật trên bảng HopDong.

**Bước 1: Kiểm tra khóa ngoại:**

IF EXISTS (

SELECT 1

FROM inserted I

LEFT JOIN KhachHang k ON i.MaKH = k.MaKH

WHERE k.MaKH IS NULL

)

BEGIN

RAISERROR (N'MaKH không tồn tại trong bảng KhachHang.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END;

- Trigger kiểm tra xem MaKH trong bảng inserted có tồn tại trong bảng KhachHang không.

- Bảng tạm inserted chứa các bản ghi mới được thêm hoặc cập nhật. Lệnh kiểm tra nếu bất kỳ MaKH nào không khớp với bảng KhachHang.

- Nếu không tồn tại thì gửi thông báo lỗi qua RAISERROR.

- Hủy giao dịch ngay lập tức với lệnh ROLLBACK TRANSACTION.

- Dừng trigger với lệnh RETURN.

**Bước 2: Kiểm tra miền giá trị tổng tiền:**

IF EXISTS (

SELECT 1

FROM inserted

WHERE TongTien < 0

)

BEGIN

RAISERROR (N'TongTien phải lớn hơn hoặc bằng 0.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END;

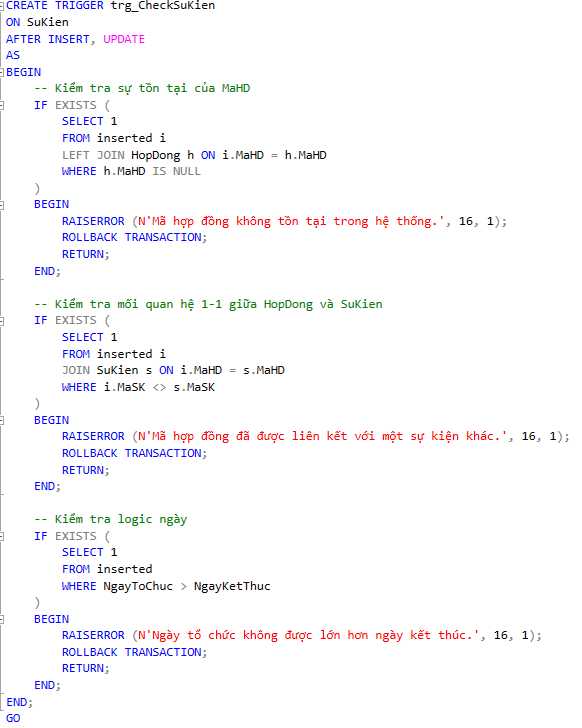
- Trigger đảm bảo rằng giá trị TongTien của các bản ghi mới không âm.

- Kiểm tra trong bảng tạm inserted nếu bất kỳ giá trị TongTien nào nhỏ hơn 0.

- Nếu TongTien âm thì gửi thông báo lỗi qua RAISERROR.

- Hủy giao dịch ngay lập tức với lệnh ROLLBACK TRANSACTION.

- Dừng trigger với lệnh RETURN.



Hình 21: Trigger kiểm tra khi thêm và cập nhập sự kiện

**Mục Đích:**  
- Trigger trg\_CheckSuKien được thiết kế để thực hiện kiểm tra dữ liệu khi có các thao tác thêm hoặc cập nhật trên bảng SuKien.

**Bước 1: Kiểm tra sự tồn tại của mã hợp đồng:**

IF EXISTS (

SELECT 1

FROM inserted I

LEFT JOIN HopDong h ON i.MaHD = h.MaHD

WHERE h.MaHD IS NULL

)

BEGIN

RAISERROR (N'Mã hợp đồng không tồn tại trong hệ thống.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END;

- Trigger đảm bảo rằng MaHD trong các bản ghi mới phải tồn tại trong bảng HopDong.

- Trigger sử dụng bảng tạm inserted để kiểm tra các bản ghi mới. Điều kiện WHERE h.MaHD IS NULL xác định các bản ghi trong inserted có MaHD không tìm thấy trong bảng HopDong.

- Nếu đã tồn tại thì gửi thông báo lỗi qua RAISERROR.

- Hủy giao dịch ngay lập tức với lệnh ROLLBACK TRANSACTION.

- Dừng trigger với lệnh RETURN.

**Bước 2: Kiểm tra mối quan hệ 1-1 giữa HopDong và SuKien:**

IF EXISTS (

SELECT 1

FROM inserted I

JOIN SuKien s ON i.MaHD = s.MaHD

WHERE i.MaSK <> s.MaSK

)

BEGIN

RAISERROR (N'Mã hợp đồng đã được liên kết với một sự kiện khác.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END;

- Trigger đảm bảo rằng mỗi mã hợp đồng (MaHD) chỉ được liên kết với một mã sự kiện (MaSK).

- Trigger sử dụng JOIN giữa bảng tạm inserted và bảng SuKien. Điều kiện WHERE i.MaSK <> s.MaSK kiểm tra nếu MaHD trong inserted đã được liên kết với một mã sự kiện khác trong SuKien.

- Nếu đã tồn tại thì gửi thông báo lỗi qua RAISERROR.

- Hủy giao dịch ngay lập tức với lệnh ROLLBACK TRANSACTION.

- Dừng trigger với lệnh RETURN.

**Bước 3: Kiểm tra logic ngày:**

IF EXISTS (

SELECT 1

FROM inserted

WHERE NgayToChuc > NgayKetThuc

)

BEGIN

RAISERROR (N'Ngày tổ chức không được lớn hơn ngày kết thúc.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END;

- Trigger đảm bảo rằng ngày tổ chức (NgayToChuc) không lớn hơn ngày kết thúc (NgayKetThuc).

- Trigger kiểm tra nếu có bất kỳ bản ghi mới nào trong bảng inserted có NgayToChuc lớn hơn NgayKetThuc.

- Nếu đã tồn tại thì gửi thông báo lỗi qua RAISERROR.

- Hủy giao dịch ngay lập tức với lệnh ROLLBACK TRANSACTION.

- Dừng trigger với lệnh RETURN.

# PHẦN 2

## Chuyển đổi mô hình ERD sang mô hình quan hệ

### Tổng quan:

* **Bước 1: Nhập và đọc file đầu vào:**

- Nhập dữ liệu từ file input.txt.

* **Bước 2: Phân tích dòng đầu vào:**

-Tách thông tin từ mỗi dòng đầu vào thành các thực thể và loại quan hệ.

* **Bước 3: Chuyển đổi mối quan hệ:**

- Chuyển đổi mối quan hệ giữa các thực thể từ mô hình (ERD) sang mô hình quan hệ.

* **Bước 4: Định dạng và chuẩn bị dữ liệu đầu ra:**

- Tạo chuỗi định dạng mô tả các thực thể và mối quan hệ sau khi chuyển đổi.

* **Bước 5: Ghi kết quả ra file:**

- Lưu kết quả chuyển đổi vào file output.txt.

### Giải thích từng thành phần chính

#### **Class Entity:**

    // Class đại diện cho thực thể hay table

    static class Entity {

        String name;

        List<String> attributes;

        List<String> primaryKeys;

        List<String> foreignKeys;

        List<String> multiValuedAttributes;

        public Entity(String name) {

            this.name = name;

            this.attributes = new ArrayList<>();

            this.primaryKeys = new ArrayList<>();

            this.foreignKeys = new ArrayList<>();

            this.multiValuedAttributes = new ArrayList<>();

        }

        public void addAttribute(String attribute) {

            attributes.add(attribute);

        }

        public void addPrimaryKey(String pk) {

            primaryKeys.add(pk);

        }

        public void addForeignKey(String fk) {

            foreignKeys.add(fk);

        }

        public void addMultiValuedAttribute(String attr) {

            multiValuedAttributes.add(attr);

        }

        public Entity copy() {

            Entity newEntity = new Entity(this.name);

            newEntity.attributes.addAll(this.attributes);

            newEntity.primaryKeys.addAll(this.primaryKeys);

            newEntity.foreignKeys.addAll(this.foreignKeys);

            newEntity.multiValuedAttributes.addAll(this.multiValuedAttributes);

            return newEntity;

        }

        public String formatEntityString() {

            // kết hợp các thuộc tính với khóa chính và khóa ngoại

            List<String> orderedAttributes = new ArrayList<>();

            orderedAttributes.addAll(primaryKeys);

            orderedAttributes.addAll(foreignKeys);

            // xóa khóa chính và khóa ngoại khỏi danh sách thuộc tính để tránh trùng lặp

            List<String> otherAttributes = new ArrayList<>(attributes);

            otherAttributes.removeAll(primaryKeys);

            otherAttributes.removeAll(foreignKeys);

            orderedAttributes.addAll(otherAttributes);

            return String.join(", ", orderedAttributes);

        }

        public String formatKeysString() {

            List<String> keyStrings = new ArrayList<>();

            for (String pk : primaryKeys) {

                keyStrings.add(pk + ": PK");

            }

            for (String fk : foreignKeys) {

                keyStrings.add(fk + ": FK");

            }

            for (String mv : multiValuedAttributes) {

                keyStrings.add(mv + ": Multivalued");

            }

            return String.join(", ", keyStrings);

        }

    }

**Mục đích:**

- Đại diện cho một thực thể (entity) trong mô hình ERD và mô hình quan hệ.

**Thuộc tính:**

* name: Tên thực thể.
* attributes: Danh sách các thuộc tính.
* primaryKeys: Danh sách các khóa chính.
* foreignKeys: Danh sách các khóa ngoại.
* multiValuedAttributes: Danh sách các thuộc tính đa trị.

**Phương thức:**

* addAttribute(String attribute): Thêm một thuộc tính.
* addPrimaryKey(String pk): Thêm một khóa chính.
* addForeignKey(String fk): Thêm một khóa ngoại.
* addMultiValuedAttribute(String attr): Thêm một thuộc tính đa trị.
* copy(): Tạo một bản sao của thực thể hiện tại.
* formatEntityString(): Định dạng chuỗi thuộc tính theo thứ tự khóa chính, khóa ngoại, rồi đến các thuộc tính khác.
* formatKeysString(): Định dạng danh sách khóa chính, khóa ngoại, và thuộc tính đa trị.

#### **Lớp Relationship:**

    // Class đại diện cho một quan hệ

    static class Relationship {

        Entity entity1;

        Entity entity2;

        RelationshipType type;

        String relationshipDescriptor;

        public Relationship(Entity entity1, Entity entity2, RelationshipType type) {

            this.entity1 = entity1;

            this.entity2 = entity2;

            this.type = type;

        }

}

**Mục đích:**

- Đại diện cho một quan hệ giữa hai thực thể.

**Thuộc tính:**

- entity1 và entity2: Hai thực thể tham gia quan hệ.

- type: Loại quan hệ (1 - n, n - n,1-1,cha-con).

- relationshipDescriptor: Chuỗi mô tả loại quan hệ (vd: "1 - n").

#### **Hàm transformRelationship**

    // chuyển đổi quan hệ thành các thực thể

    public static List<Entity> transformRelationship(Relationship relationship) {

        List<Entity> transformedEntities = new ArrayList<>();

        switch (relationship.type) {

            case ONE\_TO\_MANY:

                // thêm khóa ngoại vào bên nhiều

                Entity manyEntity = relationship.entity2.copy();

                manyEntity.addAttribute(relationship.entity1.primaryKeys.get(0));

                manyEntity.addForeignKey(relationship.entity1.primaryKeys.get(0));

                transformedEntities.add(relationship.entity1);

                transformedEntities.add(manyEntity);

                break;

            case PARENT\_CHILD:

                // xem như là 1-1 và thêm khóa ngoại vào bên con

                Entity childEntity = relationship.entity2.copy();

                childEntity.addAttribute(relationship.entity1.primaryKeys.get(0));

                childEntity.addForeignKey(relationship.entity1.primaryKeys.get(0));

                transformedEntities.add(relationship.entity1);

                transformedEntities.add(childEntity);

                break;

            case MANY\_TO\_MANY:

                // tạo thực thể trung gian

                Entity intermediaryEntity = new Entity(

                    relationship.entity1.name + "\_" + relationship.entity2.name

                );

                intermediaryEntity.addAttribute(relationship.entity1.primaryKeys.get(0));

                intermediaryEntity.addAttribute(relationship.entity2.primaryKeys.get(0));

                intermediaryEntity.addPrimaryKey(relationship.entity1.primaryKeys.get(0));

                intermediaryEntity.addPrimaryKey(relationship.entity2.primaryKeys.get(0));

                intermediaryEntity.addForeignKey(relationship.entity1.primaryKeys.get(0));

                intermediaryEntity.addForeignKey(relationship.entity2.primaryKeys.get(0));

                transformedEntities.add(relationship.entity1);

                transformedEntities.add(relationship.entity2);

                transformedEntities.add(intermediaryEntity);

                break;

            case ONE\_TO\_ONE:

                // thêm khóa ngoại vào bên thứ 2

                Entity oneToOneEntity = relationship.entity2.copy();

                oneToOneEntity.addAttribute(relationship.entity1.primaryKeys.get(0));

                oneToOneEntity.addForeignKey(relationship.entity1.primaryKeys.get(0));

                transformedEntities.add(relationship.entity1);

                transformedEntities.add(oneToOneEntity);

                break;

            case MULTI\_VALUED\_ATTRIBUTE:

                // xử lý thuộc tính đa giá trị

                Entity mainEntity = relationship.entity1.copy();

                // xóa thuộc tính đa giá trị khỏi thực thể chính

                mainEntity.attributes.removeAll(relationship.entity1.multiValuedAttributes);

                for (String mvAttr : relationship.entity1.multiValuedAttributes) {

                    //  tạo bảng riêng cho mỗi thuộc tính đa giá trị

                    Entity multiValuedEntity = new Entity(

                        relationship.entity1.name + "\_" + mvAttr

                    );

                    multiValuedEntity.addAttribute(relationship.entity1.primaryKeys.get(0));

                    multiValuedEntity.addAttribute(mvAttr);

                    multiValuedEntity.addPrimaryKey(relationship.entity1.primaryKeys.get(0));

                    multiValuedEntity.addPrimaryKey(mvAttr);

                    multiValuedEntity.addForeignKey(relationship.entity1.primaryKeys.get(0));

                    transformedEntities.add(mainEntity);

                    transformedEntities.add(multiValuedEntity);

                }

                break;

        }

        return transformedEntities;

    }

**Mục đích:**

- Hàm transformRelationship chịu trách nhiệm chuyển đổi một quan hệ trong mô hình ERD sang danh sách các thực thể trong mô hình quan hệ, phù hợp với quy tắc của từng loại quan hệ như 1 - n, n - n, 1 - 1, thuộc tính đa trị, và cha con.

**Chi tiết xử lý từng loại quan hệ:**

- Hàm nhận vào một đối tượng Relationship mô tả hai thực thể (entity1 và entity2), loại quan hệ (type), và các thuộc tính liên quan.

- Hàm sử dụng cấu trúc switch-case để xác định loại quan hệ và áp dụng các quy tắc chuyển đổi tương ứng.

* 1 - n (One-to-Many):

Thêm khóa ngoại từ thực thể "1" vào thực thể "n".

Ví dụ:

* + input.txt

[KhachHang] (MaKH, TenKH, DiaChi, SoDienThoai, Email) (MaKH: PK) - [HopDong] (MaHD, NgayKyKet, TongTien, TrangThaiHopDong) (MaHD: PK): 1 - n

* + output.txt

[KhachHang] (MaKH, TenKH, DiaChi, SoDienThoai, Email) (MaKH: PK) - [HopDong] (MaHD, NgayKyKet, TongTien, TrangThaiHopDong, MaKH) (MaHD: PK, MaKH: FK): 1 - n

* n - n (Many-to-Many):

Tạo thực thể trung gian chứa khóa chính từ cả hai thực thể.

Ví dụ:

* + input.txt

[SuKien] (MaSK, TenSK, NgayToChuc, NgayKetThuc, DiaDiem, MoTa) (MaSK: PK) - [NhanSu] (MaNS, TenNS, ChucVu, SoDienThoai, Email) (MaNS: PK): n - n

* + output.txt

[SuKien] (MaSK, TenSK, NgayToChuc, NgayKetThuc, DiaDiem, MoTa) (MaSK: PK) - [SuKien\_NhanSu] (MaSK, MaNS) (MaSK: PK, MaNS: PK, MaSK: FK, MaNS: FK): 1 - n

[NhanSu] (MaNS, TenNS, ChucVu, SoDienThoai, Email) (MaNS: PK) - [SuKien\_NhanSu] (MaSK, MaNS) (MaSK: PK, MaNS: PK, MaSK: FK, MaNS: FK): 1 - n

* 1 - 1 (One-to-One):

Thêm khóa ngoại từ một thực thể vào thực thể còn lại.

Ví dụ:

* + input.txt

[HopDong] (MaHD, NgayKyKet, TongTien, TrangThaiHopDong) (MaHD: PK) - [SuKien] (MaSK, TenSuKien, NgayToChuc, NgayKetThuc, DiaDiem, MoTa) (MaSK: PK): 1 - 1

* + output.txt

[HopDong] (MaHD, NgayKyKet, TongTien, TrangThaiHopDong) (MaHD: PK)- [SuKien] (MaSK, TenSuKien, NgayToChuc, NgayKetThuc, DiaDiem, MoTa, MaHD) (MaSK: PK, MaHD: FK): 1 - 1

* Thuộc tính đa trị (Multivalued Attribute):

Tạo thực thể riêng chứa thuộc tính đa trị và khóa chính của thực thể ban đầu.

Ví dụ:

* + input.txt

[Nhan vien] (MaNV, Ten, SoDT) (MaNV: PK, SoDT: Multivalued)

* + output.txt

[Nhan vien] (MaNV, Ten) (PK: MaNV) -[Nhanvien\_SoDT] (MaNV, SoDT) (MaNV: PK, SoDT: PK, MaNV: FK): 1 - n

* Cha - Con (Parent-Child):

Thêm khóa ngoại từ thực thể cha vào thực thể con.

Ví dụ:

* + input.txt

[DichVu] (MaDV, TenDV, DonGiaDichVu, TinhTrangDV, DonViTinh, MoTaDV) (MaDV: PK) - [Loa] (MaLoa, LoaiLoa, CongSuat, DonGiaLoa) (MaLoa: PK): Cha - Con

* + output.txt

[Nhan vien] (MaNV, Ten) (PK: MaNV) -[Nhanvien\_SoDT] (MaNV, SoDT) (MaNV: PK, SoDT: PK, MaNV: FK): 1 - n

**Quy trình hoạt động:**

- Sao chép các thực thể gốc để tránh thay đổi dữ liệu ban đầu.

- Áp dụng quy tắc chuyển đổi tùy theo loại quan hệ.

- Trả về danh sách các thực thể đã được chỉnh sửa hoặc bổ sung.

#### **Hàm processInputFile**

    // đọc từ file input và ghi kết quả vào file output

    // mỗi quan hệ được chuyển thành các thực thể và được ghi vào file output

    public static void processInputFile(String inputFile, String outputFile) throws IOException {

        // mở file input và output

        // sử dụng BufferedReader và BufferedWriter để đọc và ghi file với mã hóa UTF-8 để hỗ trợ tiếng Việt nếu có

        BufferedReader reader = new BufferedReader(new FileReader(inputFile));

        BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile));

        String line;

        while ((line = reader.readLine()) != null) {

            if (line.trim().isEmpty()) continue; // bỏ qua dòng trống

            // đọc từng dòng và chuyển thành quan hệ

            Relationship relationship = parseRelationshipLine(line); // gọi hàm parseRelationshipLine để phân tích dòng đầu vào

            // chuyển quan hệ thành các thực thể

            List<Entity> transformedEntities = transformRelationship(relationship); // gọi hàm transformRelationship để chuyển quan hệ thành các thực thể

            // ghi kết quả vào file output

            if (relationship.type == RelationshipType.MULTI\_VALUED\_ATTRIBUTE) { // xử lý thuộc tính đa giá trị

                //  thực thể chính và thực thể đa giá trị được kết nối bằng dấu -

                Entity mainEntity = transformedEntities.get(0);

                Entity multiValuedEntity = transformedEntities.get(1);

                // kết nối thực thể chính và thực thể đa giá trị

                String outputLine = String.format("[%s] (%s) (%s) -[%s] (%s) (%s): 1 - n",

                    mainEntity.name,

                    mainEntity.formatEntityString(),

                    mainEntity.formatKeysString(),

                    multiValuedEntity.name,

                    multiValuedEntity.formatEntityString(),

                    multiValuedEntity.formatKeysString()

                );

                writer.write(outputLine);

                writer.newLine();

            } else if (relationship.type == RelationshipType.MANY\_TO\_MANY) { // xử lý quan hệ nhiều - nhiều

                Entity entity1 = transformedEntities.get(0);

                Entity entity2 = transformedEntities.get(1);

                Entity intermediaryEntity = transformedEntities.get(2);

                // kết nối thực thể 1 và thực thể trung gian

                String outputLine1 = String.format("[%s] (%s) (%s) - [%s] (%s) (%s): 1 - n",

                    entity1.name,

                    entity1.formatEntityString(),

                    entity1.formatKeysString(),

                    intermediaryEntity.name,

                    intermediaryEntity.formatEntityString(),

                    intermediaryEntity.formatKeysString()

                );

                writer.write(outputLine1);

                writer.newLine();

                // kết nối thực thể 2 và thực thể trung gian

                String outputLine2 = String.format("[%s] (%s) (%s) - [%s] (%s) (%s): 1 - n",

                    entity2.name,

                    entity2.formatEntityString(),

                    entity2.formatKeysString(),

                    intermediaryEntity.name,

                    intermediaryEntity.formatEntityString(),

                    intermediaryEntity.formatKeysString()

                );

                writer.write(outputLine2);

                writer.newLine();

            } else { // xử lý các trường hợp còn lại: 1 - n, 1 - 1, cha - con

                // kết nối 2 thực thể với nhau

                Entity entity1 = transformedEntities.get(0);

                Entity entity2 = transformedEntities.get(1);

                String outputLine = String.format("[%s] (%s) (%s) - [%s] (%s) (%s): %s",

                    entity1.name,

                    entity1.formatEntityString(),

                    entity1.formatKeysString(),

                    entity2.name,

                    entity2.formatEntityString(),

                    entity2.formatKeysString(),

                    relationship.relationshipDescriptor

                );

                writer.write(outputLine);

                writer.newLine();

            }

            writer.newLine();

        }

        reader.close();

        writer.close();

    }

**Mục đích:**

- Hàm processInputFile đóng vai trò trung tâm trong chương trình, chịu trách nhiệm thực hiện các bước sau:

1. Đọc dữ liệu từ tệp đầu vào input.txt, nơi chứa các mô tả quan hệ trong mô hình ERD.
2. Phân tích từng dòng để xác định thực thể và loại quan hệ.
3. Áp dụng quy tắc chuyển đổi để chuyển từ mô hình ERD sang mô hình quan hệ.
4. Ghi kết quả đã chuyển đổi vào tệp output.txt theo định dạng chuẩn.

**Cấu trúc và quy trình hoạt động:**

**1. Đọc tệp input và output** // mở file input và output

        // sử dụng BufferedReader và BufferedWriter để đọc và ghi file

        BufferedReader reader = new BufferedReader(new FileReader(inputFile));

        BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile));

- Tệp đầu vào chứa thông tin mô tả các quan hệ trong mô hình ERD.

- Tệp đầu ra sẽ chứa mô hình quan hệ sau khi đã chuyển đổi.

**2. Đọc và xử lý từng dòng**

while ((line = reader.readLine()) != null) {

            if (line.trim().isEmpty()) continue; // bỏ qua dòng trống

- Từng dòng trong tệp input.txt được đọc và xử lý tuần tự.

- Các dòng trống hoặc không hợp lệ sẽ bị bỏ qua.

- Đảm bảo chương trình chỉ xử lý các dòng chứa thông tin hợp lệ, tránh lỗi trong quá trình phân tích.

**3. Phân tích dòng đầu vào**

            Relationship relationship = parseRelationshipLine(line); // gọi hàm parseRelationshipLine để phân tích dòng đầu vào

- Hàm parseRelationshipLine được gọi để tách thông tin từ dòng đầu vào.

- Nếu dòng chứa thuộc tính đa trị, sử dụng hàm parseMultivaluedAttribute.

- Trích xuất thông tin chi tiết như tên thực thể, danh sách thuộc tính, khóa chính, khóa ngoại, và loại quan hệ.

-> Hàm parseRelationshipLine sẽ được đề cập cụ thể sau

**4. Chuyển đổi quan hệ**

            // chuyển quan hệ thành các thực thể

            List<Entity> transformedEntities = transformRelationship(relationship); // gọi hàm transformRelationship để chuyển quan hệ thành các thực thể

- Gọi hàm transformRelationship để chuyển đổi quan hệ từ mô hình ERD sang mô hình quan hệ.

- Dựa trên loại quan hệ (1 - n, n - n, 1 - 1, v.v.), thực thể và quan hệ sẽ được điều chỉnh hoặc bổ sung.

-> Hàm transformRelationship sẽ được đề cập cụ thể sau

**5. Ghi kết quả vào tệp output**

Phân loại theo loại quan hệ:

- Đoạn mã sử dụng cấu trúc if-else để xử lý từng loại quan hệ:

+ Thuộc tính đa trị (MULTI\_VALUED\_ATTRIBUTE)

+ Quan hệ nhiều-nhiều (MANY\_TO\_MANY)

+ Các quan hệ còn lại như 1 - n, 1 - 1, hoặc cha-con.

* **Xử lý thuộc tính đa trị (MULTI\_VALUED\_ATTRIBUTE)**

if (relationship.type == RelationshipType.MULTI\_VALUED\_ATTRIBUTE) { // xử lý thuộc tính đa giá trị

                //  thực thể chính và thực thể đa giá trị được kết nối bằng dấu -

                Entity mainEntity = transformedEntities.get(0);

                Entity multiValuedEntity = transformedEntities.get(1);

                // kết nối thực thể chính và thực thể đa giá trị

                String outputLine = String.format("[%s] (%s) (%s) -[%s] (%s) (%s): 1 - n",

                    mainEntity.name,

                    mainEntity.formatEntityString(),

                    mainEntity.formatKeysString(),

                    multiValuedEntity.name,

                    multiValuedEntity.formatEntityString(),

                    multiValuedEntity.formatKeysString()

                );

                writer.write(outputLine);

                writer.newLine();

            }

- Khi thuộc tính đa trị được phát hiện, hệ thống tạo ra hai thực thể:

+ Thực thể chính: Thực thể gốc chứa các thuộc tính không đa trị.

+ Thực thể đa trị: Một bảng phụ chứa các giá trị thuộc tính đa trị và khóa chính của thực thể chính.

- Lấy thực thể chính (mainEntity) và thực thể đa trị (multiValuedEntity) từ danh sách transformedEntities.

- Định dạng chuỗi mô tả quan hệ giữa hai thực thể theo quy tắc 1 - n.

- Ghi kết quả vào tệp output.txt.

* **Xử lý quan hệ nhiều-nhiều (MANY\_TO\_MANY)**

else if (relationship.type == RelationshipType.MANY\_TO\_MANY) { // xử lý quan hệ nhiều - nhiều

                Entity entity1 = transformedEntities.get(0);

                Entity entity2 = transformedEntities.get(1);

                Entity intermediaryEntity = transformedEntities.get(2);

                // kết nối thực thể 1 và thực thể trung gian

                String outputLine1 = String.format("[%s] (%s) (%s) - [%s] (%s) (%s): 1 - n",

                    entity1.name,

                    entity1.formatEntityString(),

                    entity1.formatKeysString(),

                    intermediaryEntity.name,

                    intermediaryEntity.formatEntityString(),

                    intermediaryEntity.formatKeysString()

                );

                writer.write(outputLine1);

                writer.newLine();

                // kết nối thực thể 2 và thực thể trung gian

                String outputLine2 = String.format("[%s] (%s) (%s) - [%s] (%s) (%s): 1 - n",

                    entity2.name,

                    entity2.formatEntityString(),

                    entity2.formatKeysString(),

                    intermediaryEntity.name,

                    intermediaryEntity.formatEntityString(),

                    intermediaryEntity.formatKeysString()

                );

                writer.write(outputLine2);

                writer.newLine();

            }

- Khi phát hiện quan hệ nhiều-nhiều, chương trình tạo thêm một thực thể trung gian (bảng phụ) để liên kết hai thực thể gốc.

- Quan hệ nhiều-nhiều được chia thành hai quan hệ 1 - n.

- Lấy hai thực thể gốc (entity1, entity2) và thực thể trung gian (intermediaryEntity) từ transformedEntities.

- Tạo hai chuỗi mô tả quan hệ:

+ Quan hệ giữa thực thể thứ nhất (entity1) và thực thể trung gian.

+ Quan hệ giữa thực thể thứ hai (entity2) và thực thể trung gian.

- Ghi kết quả hai quan hệ này vào tệp output.txt.

* **Xử lý các quan hệ còn lại (1 - n, 1 - 1, cha - con):**

else { // xử lý các trường hợp còn lại: 1 - n, 1 - 1, cha - con

                // kết nối 2 thực thể với nhau

                Entity entity1 = transformedEntities.get(0);

                Entity entity2 = transformedEntities.get(1);

                String outputLine = String.format("[%s] (%s) (%s) - [%s] (%s) (%s): %s",

                    entity1.name,

                    entity1.formatEntityString(),

                    entity1.formatKeysString(),

                    entity2.name,

                    entity2.formatEntityString(),

                    entity2.formatKeysString(),

                    relationship.relationshipDescriptor

                );

                writer.write(outputLine);

                writer.newLine();

            }

- Với các loại quan hệ khác (1 - n, 1 - 1, cha-con), chương trình chỉ cần kết nối hai thực thể trực tiếp.

- Thêm thông tin về khóa chính, khóa ngoại, và mô tả loại quan hệ.

- Lấy hai thực thể từ transformedEntities.

- Định dạng chuỗi mô tả quan hệ giữa hai thực thể.

- Ghi kết quả vào tệp output.txt.

**6. Đóng tài nguyên:**

        reader.close();

        writer.close();

- Sau khi xử lý xong, đóng BufferedReader và BufferedWriter.

#### **Hàm parseRelationshipLine**

    // phân tích dòng đầu vào và trả về một quan hệ

    private static Relationship parseRelationshipLine(String line) {

        try {

            // Phân tích dòng đầu vào để trích xuất thông tin về các thực thể và quan hệ

            Pattern pattern = Pattern.compile(

                "\\[(?<entity1Name>.+?)\\]\\s\*\\((?<entity1Attrs>.+?)\\)\\s\*\\((?<entity1Keys>.+?)\\)\\s\*-\\s\*" +

                "\\[(?<entity2Name>.+?)\\]\\s\*\\((?<entity2Attrs>.+?)\\)\\s\*\\((?<entity2Keys>.+?)\\)(?::\\s\*(?<relationshipType>.+))?"

            ); // sử dụng regex để phân tích dòng đầu vào thành các nhóm tương ứng với thông tin về các thực thể và quan hệ

            Matcher matcher = pattern.matcher(line);

            if (!matcher.matches()) {

                // Xử lý thuộc tính đa giá trị

                if (line.contains(": Multivalued")) {

                    return parseMultivaluedAttribute(line);

                }

                throw new IllegalArgumentException("Invalid input format: " + line);

            }

            // Trích xuất thông tin về các thực thể và quan hệ

            String entity1Name = matcher.group("entity1Name").trim();

            String entity1Attrs = matcher.group("entity1Attrs").trim();

            String entity1Keys = matcher.group("entity1Keys").trim();

            String entity2Name = matcher.group("entity2Name").trim();

            String entity2Attrs = matcher.group("entity2Attrs").trim();

            String entity2Keys = matcher.group("entity2Keys").trim();

            String relationshipType = matcher.group("relationshipType");

            if (relationshipType == null) {

                throw new IllegalArgumentException("Relationship type missing in line: " + line);

            }

            relationshipType = relationshipType.trim();

            // Phân tích thông tin về các thực thể

            Entity entity1 = parseEntity(entity1Name, entity1Attrs, entity1Keys);

            Entity entity2 = parseEntity(entity2Name, entity2Attrs, entity2Keys);

            // Xác định loại quan hệ

            RelationshipType type;

            if (relationshipType.equalsIgnoreCase("1 - n")) {

                type = RelationshipType.ONE\_TO\_MANY;

            } else if (relationshipType.equalsIgnoreCase("n - n")) {

                type = RelationshipType.MANY\_TO\_MANY;

            } else if (relationshipType.equalsIgnoreCase("1 - 1")) {

                type = RelationshipType.ONE\_TO\_ONE;

            } else if (relationshipType.equalsIgnoreCase("Cha - Con")) {

                type = RelationshipType.PARENT\_CHILD;

            } else {

                throw new IllegalArgumentException("Không nhận ra mối quan hệ: " + relationshipType);

            }

            // Tạo quan hệ

            Relationship relationship = new Relationship(entity1, entity2, type);

            relationship.relationshipDescriptor = type == RelationshipType.PARENT\_CHILD ? "1 - 1" : relationshipType;

            return relationship;

        } catch (Exception e) {

            throw new IllegalArgumentException("Invalid input format: " + line, e);

        }

}

- Hàm parseRelationshipLine phân tích từng dòng đầu vào trong tệp input.txt để tách thông tin về các thực thể và loại quan hệ.

**Mục đích**

- Phân tích chuỗi mô tả quan hệ trong mô hình ERD để trích xuất:

+ Tên của hai thực thể.

+ Danh sách thuộc tính của từng thực thể.

+ Danh sách khóa chính và khóa ngoại của từng thực thể.

+ Loại quan hệ (1 - n, n - n, 1 - 1, hoặc Cha - Con).

- Trả về một đối tượng Relationship chứa thông tin đã được phân tích.

**Cấu trúc và quy trình hoạt động:**

1. **Phân tích dòng đầu vào bằng Regular Expression**

// Phân tích dòng đầu vào để trích xuất thông tin về các thực thể và quan hệ

            Pattern pattern = Pattern.compile(

                "\\[(?<entity1Name>.+?)\\]\\s\*\\((?<entity1Attrs>.+?)\\)\\s\*\\((?<entity1Keys>.+?)\\)\\s\*-\\s\*" +

                "\\[(?<entity2Name>.+?)\\]\\s\*\\((?<entity2Attrs>.+?)\\)\\s\*\\((?<entity2Keys>.+?)\\)(?::\\s\*(?<relationshipType>.+))?"

            ); // sử dụng regex để phân tích dòng đầu vào thành các nhóm tương ứng với thông tin về các thực thể và quan hệ

            Matcher matcher = pattern.matcher(line);

- Sử dụng biểu thức chính quy (regex) để phân tích cấu trúc của dòng đầu vào.

- Regex được thiết kế để tách thông tin theo định dạng:

[Tên thực thể 1] (Danh sách thuộc tính) (Danh sách khóa) - [Tên thực thể 2] (Danh sách thuộc tính) (Danh sách khóa): Loại quan hệ

**Ý nghĩa:**

Nhóm 1: entity1Name – Tên của thực thể thứ nhất.

Nhóm 2: entity1Attrs – Danh sách thuộc tính của thực thể thứ nhất.

Nhóm 3: entity1Keys – Danh sách khóa của thực thể thứ nhất.

Nhóm 4: entity2Name – Tên của thực thể thứ hai.

Nhóm 5: entity2Attrs – Danh sách thuộc tính của thực thể thứ hai.

Nhóm 6: entity2Keys – Danh sách khóa của thực thể thứ hai.

Nhóm 7: relationshipType – Loại quan hệ giữa hai thực thể.

**2. Kiểm tra và xử lý thuộc tính đa trị:**

            if (!matcher.matches()) {

                // Xử lý thuộc tính đa giá trị

                if (line.contains(": Multivalued")) {

                    return parseMultivaluedAttribute(line);

                }

                throw new IllegalArgumentException("Invalid input format: " + line);

            }

- Nếu dòng đầu vào không khớp với regex, kiểm tra xem đây có phải là mô tả thuộc tính đa trị hay không.

- Sử dụng hàm parseMultivaluedAttribute (sẽ được đề cập chi tiết sau) để xử lý thuộc tính đa trị.

- Xử lý riêng thuộc tính đa trị, vốn không theo định dạng chung của các quan hệ khác.

**3. Trích xuất thông tin về thực thể và quan hệ**

            // Trích xuất thông tin về các thực thể và quan hệ

            String entity1Name = matcher.group("entity1Name").trim();

            String entity1Attrs = matcher.group("entity1Attrs").trim();

            String entity1Keys = matcher.group("entity1Keys").trim();

            String entity2Name = matcher.group("entity2Name").trim();

            String entity2Attrs = matcher.group("entity2Attrs").trim();

            String entity2Keys = matcher.group("entity2Keys").trim();

            String relationshipType = matcher.group("relationshipType");

            if (relationshipType == null) {

                throw new IllegalArgumentException("Relationship type missing in line: " + line);

            }

            relationshipType = relationshipType.trim();

- Nếu dòng đầu vào khớp với regex, trích xuất từng phần thông tin:

+ Tên thực thể.

+ Danh sách thuộc tính.

+ Danh sách khóa.

+ Loại quan hệ.

- Tách từng phần thông tin cần thiết từ dòng đầu vào để chuẩn bị cho quá trình chuyển đổi.

**4. Phân tích thực thể:**

            // Phân tích thông tin về các thực thể

            Entity entity1 = parseEntity(entity1Name, entity1Attrs, entity1Keys);

            Entity entity2 = parseEntity(entity2Name, entity2Attrs, entity2Keys);

- Sử dụng hàm parseEntity (sẽ được đề cập chi tiết sau) để chuyển thông tin tên, thuộc tính, và khóa của mỗi thực thể thành đối tượng Entity.

- Chuẩn hóa thông tin của mỗi thực thể để dễ dàng sử dụng trong các bước tiếp theo.

**5. Xác định loại quan hệ:**

            // Xác định loại quan hệ

            RelationshipType type;

            if (relationshipType.equalsIgnoreCase("1 - n")) {

                type = RelationshipType.ONE\_TO\_MANY;

            } else if (relationshipType.equalsIgnoreCase("n - n")) {

                type = RelationshipType.MANY\_TO\_MANY;

            } else if (relationshipType.equalsIgnoreCase("1 - 1")) {

                type = RelationshipType.ONE\_TO\_ONE;

            } else if (relationshipType.equalsIgnoreCase("Cha - Con")) {

                type = RelationshipType.PARENT\_CHILD;

            } else {

                throw new IllegalArgumentException("Không nhận ra mối quan hệ: " + relationshipType);

            }

- So sánh chuỗi relationshipType với các loại quan hệ được hỗ trợ để xác định loại quan hệ.

- Chuyển chuỗi mô tả loại quan hệ thành một đối tượng RelationshipType để sử dụng trong các bước tiếp theo.

**6. Tạo đối tượng Relationship:**

            // Tạo quan hệ

            Relationship relationship = new Relationship(entity1, entity2, type);

            relationship.relationshipDescriptor = type == RelationshipType.PARENT\_CHILD ? "1 - 1" : relationshipType;

            return relationship;

- Sử dụng thông tin thực thể và loại quan hệ để tạo một đối tượng Relationship.

- Đóng gói toàn bộ thông tin về hai thực thể và loại quan hệ để trả về cho các bước xử lý tiếp theo.

#### **Hàm parseMultivaluedAttribute**

    // phân tích thuộc tính đa giá trị từ chuỗi

    private static Relationship parseMultivaluedAttribute(String line) {

        // Xử lý thuộc tính đa giá trị

        Pattern pattern = Pattern.compile("\\[(.+?)\\]\\s\*\\((.+?)\\)\\s\*\\((.+?)\\)");

        Matcher matcher = pattern.matcher(line);

        if (!matcher.find()) {

            throw new IllegalArgumentException("Invalid input format for multivalued attribute: " + line);

        }

        String entityName = matcher.group(1);

        String attributesStr = matcher.group(2);

        String keysStr = matcher.group(3);

        Entity entity = parseEntity(entityName, attributesStr, keysStr);

        // Tạo quan hệ

        Relationship relationship = new Relationship(entity, null, RelationshipType.MULTI\_VALUED\_ATTRIBUTE);

        relationship.relationshipDescriptor = "1 - n";

        return relationship;

    }

- Hàm parseMultivaluedAttribute xử lý các thuộc tính đa trị trong mô hình ERD. Những thuộc tính này không được lưu trực tiếp trong bảng chính mà phải được chuyển đổi thành một bảng phụ.

**Mục đích**

- Phân tích một dòng mô tả thuộc tính đa trị từ tệp đầu vào.

- Tách thông tin thực thể, thuộc tính và khóa.

- Tạo một đối tượng Relationship để chỉ ra rằng thực thể này có thuộc tính đa trị.

**Cấu trúc và quy trình hoạt động:**

1. **Kiểm tra định dạng dòng đầu vào:**

        // Xử lý thuộc tính đa giá trị

        Pattern pattern = Pattern.compile("\\[(.+?)\\]\\s\*\\((.+?)\\)\\s\*\\((.+?)\\)");

        Matcher matcher = pattern.matcher(line);

        if (!matcher.find()) {

            throw new IllegalArgumentException("Invalid input format for multivalued attribute: " + line);

        }

- Sử dụng regex để kiểm tra xem dòng đầu vào có đúng định dạng thuộc tính đa trị hay không.

- Giải thích:

* \\[(.+?)\\]: Tên thực thể nằm trong dấu ngoặc vuông [ ].
* \\((.+?)\\): Danh sách thuộc tính nằm trong ngoặc đơn ( ).
* \\((.+?)\\): Danh sách khóa nằm trong ngoặc đơn ( ).

1. **Trích xuất thông tin từ dòng đầu vào:**

        String entityName = matcher.group(1);

        String attributesStr = matcher.group(2);

        String keysStr = matcher.group(3);

- Sau khi kiểm tra dòng đầu vào hợp lệ, regex sẽ tách thông tin thành ba phần:

+ entityName: Tên thực thể.

+ attributesStr: Danh sách thuộc tính.

+ keysStr: Danh sách khóa.

1. **Tạo đối tượng Entity:**

        Entity entity = parseEntity(entityName, attributesStr, keysStr);

- Sử dụng hàm parseEntity (sẽ được đề cập chi tiết sau) để chuyển thông tin entityName, attributesStr, và keysStr thành một đối tượng Entity.

-Tách riêng thông tin về thực thể để chuẩn bị xử lý thuộc tính đa trị.

**4. Tạo đối tượng Relationship**

        // Tạo quan hệ

        Relationship relationship = new Relationship(entity, null, RelationshipType.MULTI\_VALUED\_ATTRIBUTE);

        relationship.relationshipDescriptor = "1 - n";

        return relationship;

- Tạo một đối tượng Relationship để chỉ ra rằng thực thể này có thuộc tính đa trị.

+ *entity*: Thực thể gốc có thuộc tính đa trị.

+ *null*: Không có thực thể thứ hai vì đây không phải là quan hệ giữa hai thực thể.

+ *RelationshipType.MULTI\_VALUED\_ATTRIBUTE*: Loại quan hệ là thuộc tính đa trị.

+ *relationshipDescriptor = "1 - n"*: Mô tả rằng thuộc tính đa trị sẽ được chuyển thành một quan hệ 1 - n.

#### **Hàm parseEntity**

    // phân tích một thực thể từ chuỗi

    private static Entity parseEntity(String entityName, String attributesStr, String keysStr) {

        Entity entity = new Entity(entityName);

        String[] attributes = attributesStr.split(",\\s\*");

        for (String attr : attributes) entity.addAttribute(attr.trim());

        // Xác định khóa chính, khóa ngoại và thuộc tính đa giá trị

        String[] keys = keysStr.split(",\\s\*");

        for (String key : keys) {

            if (key.contains(": PK")) {

                entity.addPrimaryKey(key.replace(": PK", "").trim());

            } else if (key.contains(": FK")) {

                entity.addForeignKey(key.replace(": FK", "").trim());

            } else if (key.contains(": Multivalued")) {

                entity.addMultiValuedAttribute(key.replace(": Multivalued", "").trim());

            }

        }

        return entity;

    }

- Hàm parseEntity chuyển đổi thông tin mô tả về một thực thể trong dòng đầu vào thành một đối tượng Entity.

**Mục Đích**

- Phân tích tên thực thể, danh sách thuộc tính, và danh sách khóa từ chuỗi đầu vào.

- Tạo một đối tượng Entity chứa thông tin chi tiết:

+ Tên thực thể.

+ Danh sách thuộc tính.

+ Danh sách khóa chính.

+ Danh sách khóa ngoại.

+ Thuộc tính đa trị (nếu có).

**Cấu trúc và quy trình xử lý:**

**1. Khởi tạo đối tượng Entity**

        Entity entity = new Entity(entityName);

- Khởi tạo một đối tượng Entity với tên được cung cấp từ tham số entityName.

**2. Phân tích danh sách thuộc tính**

        String[] attributes = attributesStr.split(",\\s\*");

        for (String attr : attributes) entity.addAttribute(attr.trim());

- Tách chuỗi attributesStr (danh sách thuộc tính) bằng dấu phẩy , và loại bỏ khoảng trắng thừa.

- Thêm từng thuộc tính vào đối tượng Entity.

**3. Phân tích danh sách khóa**

        // Xác định khóa chính, khóa ngoại và thuộc tính đa giá trị

        String[] keys = keysStr.split(",\\s\*");

        for (String key : keys) {

            if (key.contains(": PK")) {

                entity.addPrimaryKey(key.replace(": PK", "").trim());

            } else if (key.contains(": FK")) {

                entity.addForeignKey(key.replace(": FK", "").trim());

            } else if (key.contains(": Multivalued")) {

                entity.addMultiValuedAttribute(key.replace(": Multivalued", "").trim());

            }

        }

- Tách chuỗi keysStr (danh sách khóa) bằng dấu phẩy ‘,’

- Phân loại khóa dựa trên nhãn:

+ PK – Khóa chính.

+ FK – Khóa ngoại.

+ Multivalued – Thuộc tính đa trị.

* PK:

- Loại bỏ nhãn : PK để lấy tên khóa chính.

- Thêm tên khóa vào danh sách khóa chính của Entity bằng addPrimaryKey.

* FK:

- Loại bỏ nhãn : FK để lấy tên khóa ngoại.

- Thêm tên khóa vào danh sách khóa ngoại của Entity bằng addForeignKey.

* Multivalued:

- Loại bỏ nhãn : Multivalued để lấy tên thuộc tính đa trị.

- Thêm thuộc tính vào danh sách thuộc tính đa trị của Entity bằng addMultiValuedAttribute.

**4. Trả về đối tượng Entity**

        return entity;

- Sau khi phân tích xong danh sách thuộc tính và danh sách khóa, trả về đối tượng Entity.

### Tóm tắt các bước chương trình thực hiện:

**1. Đọc tệp đầu vào**

- Hàm: processInputFile

- Mục đích:

+ Đọc tệp input.txt chứa thông tin về các thực thể và quan hệ trong mô hình ERD.

+ Kiểm tra từng dòng và bỏ qua các dòng trống.

- Kết quả: Đưa từng dòng dữ liệu đến bước phân tích.

**2. Phân tích dòng đầu vào**

- Hàm: parseRelationshipLine

- Mục đích:

+ Phân tích từng dòng để trích xuất thông tin:

* Tên thực thể.
* Danh sách thuộc tính.
* Danh sách khóa chính, khóa ngoại, hoặc thuộc tính đa trị.
* Loại quan hệ (1 - n, n - n, 1 - 1, Cha - Con, hoặc thuộc tính đa trị).

+ Nếu dòng mô tả thuộc tính đa trị, chuyển sang hàm parseMultivaluedAttribute.

- Kết quả: Trả về đối tượng Relationship mô tả quan hệ giữa các thực thể.

**3. Phân tích thông tin thực thể**

- Hàm: parseEntity

- Mục đích:

+ Chuyển thông tin từ chuỗi tên thực thể, thuộc tính, và khóa thành đối tượng Entity.

+ Phân loại khóa chính, khóa ngoại, và thuộc tính đa trị.

- Kết quả: Trả về đối tượng Entity chứa các thông tin chuẩn hóa về thực thể.

**4. Xử lý thuộc tính đa trị**

- Hàm: parseMultivaluedAttribute

- Mục đích:

+ Tách thông tin về thực thể có thuộc tính đa trị.

+ Tạo một đối tượng Relationship để chỉ ra rằng thực thể này có thuộc tính đa trị.

- Kết quả: Trả về quan hệ loại MULTI\_VALUED\_ATTRIBUTE giữa thực thể và bảng phụ lưu giá trị thuộc tính đa trị.

**5. Chuyển đổi quan hệ**

- Hàm: transformRelationship

- Mục đích:

+ Dựa vào loại quan hệ, áp dụng quy tắc chuyển đổi:

* 1 - n: Thêm khóa ngoại vào thực thể "n".
* n - n: Tạo một bảng phụ (thực thể trung gian) để lưu khóa chính của cả hai thực thể.
* 1 - 1: Thêm khóa ngoại từ thực thể thứ nhất vào thực thể thứ hai.
* Cha - Con: Thêm khóa ngoại từ thực thể cha vào thực thể con.
* Thuộc tính đa trị: Tạo bảng phụ lưu giá trị thuộc tính đa trị.

- Kết quả: Trả về danh sách các thực thể đã được sửa đổi hoặc bổ sung.

**6. Ghi kết quả ra tệp**

- Hàm: processInputFile

- Mục đích:

+ Ghi kết quả chuyển đổi từ mô hình ERD sang mô hình quan hệ vào tệp output.txt.

+ Định dạng kết quả tùy theo loại quan hệ:

* 1 - n: Kết nối hai thực thể với thông tin khóa chính và khóa ngoại.
* n - n: Ghi hai quan hệ 1 - n giữa bảng trung gian và mỗi thực thể gốc.
* 1 - 1: Ghi kết nối hai thực thể với thông tin khóa ngoại.
* Thuộc tính đa trị: Ghi kết nối giữa bảng phụ và bảng chính.

- Kết quả: Tệp output.txt chứa mô hình quan hệ dạng bảng.

### Hình ảnh các testcase:

#### **Quan hệ 1 - n**



Hình 22: Testcase input cho quan hệ 1 - n



Hình 23: Testcase output cho quan hệ 1 - n

#### **Quan hệ n - n**



Hình 24: Testcase input cho quan hệ n - n



Hình 25: Testcase output cho quan hệ n - n

#### **Quan hệ 1 - 1**



Hình 26: Testcase input cho quan hệ 1 - 1



Hình 27: Testcase output cho quan hệ 1 - 1

#### **Thuộc tính đa trị**



Hình 28: Testcase input cho thuộc tính đa trị



Hình 29: Testcase output cho thuộc tính đa trị

#### **Quan hệ cha - con**



Hình 30: Testcase cho input quan hệ cha - con



Hình 31: Testcase cho output quan hệ cha - con

## Tìm khóa của lược đồ cơ sở dữ liệu:

### Tổng quan:

- Chương trình sử dụng thuật toán liệt kê tập con bằng mã nhị phân để tìm tất cả các khóa có thể của lược đồ.

- Cụ thể, thuật toán dựa vào việc:

- Sinh tất cả các tập con của tập thuộc tính:

+ Dựa trên bitmask, chương trình duyệt qua mọi tổ hợp thuộc tính để kiểm tra xem tập con đó có phải là một khóa hay không.

- Kiểm tra tính chất khóa của mỗi tập con:

+ Tính bao đóng (closure) của tập con.

+ Nếu bao đóng chứa tất cả các thuộc tính trong lược đồ, tập con đó được xem là một khóa.

**Bước 1: Đọc input từ file:**

- Đọc và phân tích các thuộc tính và phụ thuộc hàm để lưu trữ trong danh sách.

**Bước 2: Tính bao đóng của thuộc tính:**

- Tính tất cả các thuộc tính có thể suy diễn được từ tập con đầu vào.

- Sử dụng vòng lặp để kiểm tra mọi phụ thuộc hàm và cập nhật bao đóng cho đến khi không còn thuộc tính nào mới được thêm vào.

**Bước 3: Tìm tất cả khóa:**

- Duyệt qua mọi tổ hợp thuộc tính (dựa trên bitmask).

- Với mỗi tập con, tính bao đóng và kiểm tra nếu bao đóng chứa tất cả các thuộc tính trong lược đồ.

- Lưu lại tất cả các khóa tìm được.

**Bước 4: Ghi kết quả ra file:**

-Ghi tổng số khóa và danh sách các khóa vào tệp đầu ra.

### Giải thích các thành phần chính:

#### **Lớp KeyFinder:**

**1. Thuộc tính:**

    // Lưu trữ các thuộc tính của lược đồ

    private List<String> attributes;

    // Lưu trữ các phụ thuộc hàm

    private List<FunctionalDependency> dependencies;

- *attributes*: lưu danh sách tất cả các thuộc tính trong lược đồ cơ sở dữ liệu.

- *dependencies*: lưu danh sách các phụ thuộc hàm được phân tích từ input.

**2. Các phương thức chính:**

* readInput(String inputFile)
* parseAttributes(String line)
* parseDependency(String line)
* closure(Set<String> initialSet)
* findAllKeys()
* writeAllKeys(String outputFile)

#### **Lớp con FunctionalDependency**

    // Lớp biểu diễn phụ thuộc hàm

    private class FunctionalDependency {

        Set<String> left;  // Các thuộc tính bên trái

        Set<String> right; // Các thuộc tính bên phải

        FunctionalDependency(Set<String> left, Set<String> right) {

            this.left = left;

            this.right = right;

        }

    }

- Lớp này biểu diễn một phụ thuộc hàm dưới dạng:

+ *left*: Tập thuộc tính bên trái.

+ *right*: Tập thuộc tính bên phải.

#### **Phương thức readInput:**

    // Đọc input từ file

    public void readInput(String inputFile) throws IOException {

        try (BufferedReader br = new BufferedReader(new FileReader(inputFile))) {

            // Đọc dòng đầu tiên chứa các thuộc tính

            String attributeLine = br.readLine();

            parseAttributes(attributeLine);

            // Đọc các phụ thuộc hàm

            String line;

            while ((line = br.readLine()) != null) {

                parseDependency(line);

            }

        }

    }

**Mục đích:**

- Tập thuộc tính được lưu vào danh sách attributes.

- Tập phụ thuộc hàm được lưu vào danh sách dependencies.

**Cấu trúc và quy trình xử lý:**

1. **Đọc dòng đầu tiên: Tập thuộc tính**

            // Đọc dòng đầu tiên chứa các thuộc tính

            String attributeLine = br.readLine();

            parseAttributes(attributeLine);

- Hàm br.readLine() đọc dòng đầu tiên trong file, lưu vào attributeLine.

- Gọi phương thức parseAttributes(attributeLine) để phân tích và lưu tập thuộc tính vào danh sách attributes.

-> phương thức parseAttributes sẽ được đề cập chi tiết sau

1. **Đọc các dòng còn lại: Tập phụ thuộc hàm**

            // Đọc các phụ thuộc hàm

            String line;

            while ((line = br.readLine()) != null) {

                parseDependency(line);

            }

- Vòng lặp while đọc từng dòng tiếp theo trong file.

- Với mỗi dòng, gọi phương thức parseDependency(line) để phân tích và lưu thông tin phụ thuộc hàm vào danh sách dependencies.

-> phương thức parseDependencysẽ được đề cập chi tiết sau

#### **Phương thức parseAttributes:**

    // Phân tích chuỗi thuộc tính

    private void parseAttributes(String line) {

        // Loại bỏ các ký tự không cần thiết và tách các thuộc tính

        line = line.replaceAll("[R()]", "").trim();

        String[] attrs = line.split(",\\s\*");

        attributes.addAll(Arrays.asList(attrs));

    }

- Phương thức parseAttributes thuộc lớp KeyFinder, phân tích chuỗi chứa danh sách các thuộc tính của lược đồ cơ sở dữ liệu và lưu chúng vào danh sách attributes.

**Mục đích**

- Phân tích chuỗi đầu vào: Chuỗi chứa tập thuộc tính thường đi kèm các ký tự không cần thiết (như R()), cần được loại bỏ.

- Lưu trữ thuộc tính: Sau khi phân tích, các thuộc tính sẽ được lưu vào danh sách attributes.

**Cấu trúc và quy trình xử lý:**

1. **Loại bỏ ký tự không cần thiết:**

line = line.replaceAll("[R()]", "").trim();

- Sử dụng phương thức replaceAll để loại bỏ các ký tự không cần thiết:

- trim(): Loại bỏ khoảng trắng thừa ở đầu và cuối chuỗi.

1. **Tách chuỗi thành danh sách thuộc tính**

String[] attrs = line.split(",\\s\*");

- Sử dụng phương thức split với biểu thức chính quy ",\\s\*" để tách các thuộc tính bằng dấu phẩy.

1. **Thêm các thuộc tính vào danh sách attributes**

attributes.addAll(Arrays.asList(attrs));

- Sử dụng Arrays.asList(attrs) để chuyển mảng chuỗi (attrs) thành danh sách.

- Thêm toàn bộ các thuộc tính từ danh sách vừa tạo vào danh sách attributes

#### **Phương thức parseDependency:**

    // Phân tích phụ thuộc hàm

    private void parseDependency(String line) {

        String[] parts = line.split("->");

        if (parts.length != 2) return; // Bỏ qua nếu không đúng định dạng

        // Phân tích các thuộc tính bên trái

        Set<String> leftAttributes = new HashSet<>();

        for (char c : parts[0].trim().toCharArray()) {

            leftAttributes.add(String.valueOf(c));

        }

        // Phân tích các thuộc tính bên phải

        Set<String> rightAttributes = new HashSet<>();

        for (char c : parts[1].trim().toCharArray()) {

            rightAttributes.add(String.valueOf(c));

        }

        // Thêm phụ thuộc hàm vào danh sách

        dependencies.add(new FunctionalDependency(leftAttributes, rightAttributes));

    }

- Phương thức parseDependency phân tích một chuỗi chứa phụ thuộc hàm và lưu thông tin vào danh sách dependencies.

**Mục đích**

- Phân tích chuỗi chứa phụ thuộc hàm: Chuỗi này được tách thành hai phần:

+ left: Tập thuộc tính bên trái mũi tên.

+ right: Tập thuộc tính bên phải mũi tên.

- Lưu trữ phụ thuộc hàm: Sau khi phân tích, thông tin được lưu vào danh sách dependencies dưới dạng đối tượng FunctionalDependency.

**Cấu trúc và quy trình xử lý:**

1. **Tách chuỗi phụ thuộc hàm:**

        String[] parts = line.split("->");

        if (parts.length != 2) return; // Bỏ qua nếu không đúng định dạng

- Sử dụng split("->") để tách chuỗi phụ thuộc hàm thành hai phần:

+ parts[0]: Thuộc tính bên trái mũi tên.

+ parts[1]: Thuộc tính bên phải mũi tên.

- Nếu chuỗi không chứa đúng một mũi tên (->), nó sẽ bị bỏ qua.

1. **Phân tích thuộc tính bên trái**

        // Phân tích các thuộc tính bên trái

        Set<String> leftAttributes = new HashSet<>();

        for (char c : parts[0].trim().toCharArray()) {

            leftAttributes.add(String.valueOf(c));

        }

- Loại bỏ khoảng trắng ở đầu và cuối chuỗi bên trái (trim()).

- Chuyển chuỗi bên trái thành từng ký tự (toCharArray()).

- Thêm từng ký tự vào tập leftAttributes.

1. **Phân tích thuộc tính bên phải**

        // Phân tích các thuộc tính bên phải

        Set<String> rightAttributes = new HashSet<>();

        for (char c : parts[1].trim().toCharArray()) {

            rightAttributes.add(String.valueOf(c));

        }

- Tương tự bước phân tích bên trái, chuỗi bên phải cũng được chuyển thành tập thuộc tính rightAttributes

1. **Lưu phụ thuộc hàm vào danh sách**

// Thêm phụ thuộc hàm vào danh sách

dependencies.add(new FunctionalDependency(leftAttributes, rightAttributes));

- Tạo một đối tượng FunctionalDependency.

- Thêm đối tượng này vào danh sách dependencies.

#### **Phương thức closure:**

    // Tính bao đóng của một tập thuộc tính

    private Set<String> closure(Set<String> initialSet) {

        Set<String> closure = new HashSet<>(initialSet);

        boolean changed;  // Biến kiểm tra xem tập thuộc tính đã thay đổi chưa

        // vòng lặp thực hiện cho đến khi không thể thêm thuộc tính mới vào tập

        do {

            changed = false;

            for (FunctionalDependency fd : dependencies) {

                if (closure.containsAll(fd.left)) {

                    for (String attr : fd.right) {

                        if (!closure.contains(attr)) {

                            closure.add(attr);

                            changed = true;

                        }

                    }

                }

            }

        } while (changed);

        return closure;

    }

- Phương thức closure được sử dụng để tính bao đóng của một tập thuộc tính trong lược đồ cơ sở dữ liệu.

**Mục đích**

- Tính bao đóng: Dựa trên tập phụ thuộc hàm, xác định tất cả các thuộc tính mà một tập thuộc tính có thể suy diễn được.

- Bao đóng giúp kiểm tra xem tập thuộc tính ban đầu có thể xác định duy nhất toàn bộ các thuộc tính khác trong lược đồ hay không.

**Cấu trúc và quy trình xử lý:**

1. **Khởi tạo bao đóng:**

        Set<String> closure = new HashSet<>(initialSet);

- Sao chép các thuộc tính từ tập đầu vào initialSet vào tập bao đóng closure.

- Ban đầu, tập bao đóng chỉ chứa các thuộc tính trong initialSet.

1. **Lặp qua các phụ thuộc hàm:**

        boolean changed;  // Biến kiểm tra xem tập thuộc tính đã thay đổi chưa

        // vòng lặp thực hiện cho đến khi không thể thêm thuộc tính mới vào tập

        do {

            changed = false;

            for (FunctionalDependency fd : dependencies) {

                if (closure.containsAll(fd.left)) {

                    for (String attr : fd.right) {

                        if (!closure.contains(attr)) {

                            closure.add(attr);

                            changed = true;

                        }

                    }

                }

            }

        } while (changed);

- Sử dụng vòng lặp để kiểm tra từng phụ thuộc hàm fd trong dependencies.

- Nếu tất cả các thuộc tính bên trái (fd.left) nằm trong bao đóng, thì các thuộc tính bên phải (fd.right) có thể được suy diễn và thêm vào bao đóng.

* if (closure.containsAll(fd.left)):

- Kiểm tra nếu tập closure chứa tất cả các thuộc tính trong fd.left (bên trái phụ thuộc hàm).

- Nếu đúng, điều này nghĩa là các thuộc tính bên phải (fd.right) có thể được suy diễn.

* if (!closure.contains(attr)):

- Kiểm tra nếu thuộc tính bên phải chưa có trong bao đóng.

- Nếu thuộc tính mới được thêm vào closure, đặt changed = true để lặp lại vòng lặp và kiểm tra các phụ thuộc khác.

* Lặp lại cho đến khi không thêm được thuộc tính mới:

- Vòng lặp do-while đảm bảo chương trình tiếp tục kiểm tra các phụ thuộc cho đến khi bao đóng không thay đổi nữa (changed = false).

1. **Trả về bao đóng:**

        return closure;

- Trả về tập bao đóng cuối cùng sau khi không còn thuộc tính nào mới có thể được suy diễn.

#### **Phương thức findAllKeys**

    // Tìm tất cả các khóa của lược đồ (không lọc tối thiểu)

    public Set<Set<String>> findAllKeys() {

        Set<Set<String>> allKeys = new HashSet<>();

        int n = attributes.size();

        // Duyệt qua tất cả các bitmask (2^n tổ hợp)

        for (int mask = 1; mask < (1 << n); mask++) {

            Set<String> subset = new HashSet<>();

            // Tạo tập con dựa trên bitmask

            for (int i = 0; i < n; i++) {

                if ((mask & (1 << i)) != 0) {

                    subset.add(attributes.get(i));

                }

            }

            // Kiểm tra nếu tập con này là một khóa

            if (closure(subset).containsAll(attributes)) {

                allKeys.add(subset); // Thêm vào tất cả các khóa

            }

        }

        return allKeys;

    }

- Phương thức findAllKeys tìm tất cả các khóa (candidate keys) của một lược đồ cơ sở dữ liệu dựa trên tập thuộc tính và tập phụ thuộc hàm.

**Mục đích**

- Tìm tất cả các khóa: Liệt kê tất cả các tập con của tập thuộc tính và kiểm tra tập con nào có thể xác định duy nhất toàn bộ lược đồ.

**Cấu trúc và quy trình xử lý:**

1. **Khởi tạo**

        Set<Set<String>> allKeys = new HashSet<>();

        int n = attributes.size();

- allKeys: Tập chứa tất cả các khóa tìm được.

- n = attributes.size(): Tổng số thuộc tính trong lược đồ, dùng để tính số tổ hợp

1. **Sinh tất cả các tập con (bitmask)**

        // Duyệt qua tất cả các bitmask (2^n tổ hợp)

        for (int mask = 1; mask < (1 << n); mask++) {

            Set<String> subset = new HashSet<>();

            // Tạo tập con dựa trên bitmask

            for (int i = 0; i < n; i++) {

                if ((mask & (1 << i)) != 0) {

                    subset.add(attributes.get(i));

                }

            }

            // Kiểm tra nếu tập con này là một khóa

            if (closure(subset).containsAll(attributes)) {

                allKeys.add(subset); // Thêm vào tất cả các khóa

            }

        }

- mask: Đại diện cho một tổ hợp (tập con) các thuộc tính bằng cách sử dụng bitmask.

- Vòng lặp con (for (int i = 0; i < n; i++)):

+ Kiểm tra bit thứ i trong mask:

Nếu bit thứ i được bật ((mask & (1 << i)) != 0), thêm thuộc tính tương ứng vào tập con subset.

- Giải thích bitmask:

+ Mỗi số nhị phân từ 1 đến đại diện cho một tập con khác nhau.

+ Số lượng tổ hợp là , trong đó bit thứ i chỉ trạng thái "chọn" hoặc "bỏ qua" thuộc tính thứ i.

- Ví dụ: attributes = [A, B, C] (n=3):

mask = 001 -> {C}

mask = 011 -> {B, C}

mask = 111 -> {A, B, C}

1. **Kiểm tra bao đóng**

            // Kiểm tra nếu tập con này là một khóa

            if (closure(subset).containsAll(attributes)) {

                allKeys.add(subset); // Thêm vào tất cả các khóa

            }

- Tính bao đóng của tập con subset bằng phương thức closure.

- Kiểm tra nếu bao đóng của subset chứa tất cả các thuộc tính (attributes):

Nếu đúng, subset là một khóa và được thêm vào allKeys.

1. **Trả về tất cả các khóa**

        return allKeys;

- Trả về danh sách allKeys chứa tất cả các tập thuộc tính được xác định là khóa.

#### **Phương thức writeAllKeys**

    // Ghi tất cả các khóa ra file

    public void writeAllKeys(String outputFile) throws IOException {

        Set<Set<String>> allKeys = findAllKeys();

        try (BufferedWriter bw = new BufferedWriter(new FileWriter(outputFile))) {

            bw.write("Tổng số khóa tìm được: " + allKeys.size());

            bw.newLine();

            for (Set<String> key : allKeys) {

                bw.write(key.toString());

                bw.newLine();

            }

        }

    }

- Phương thức writeAllKeys tìm tất cả các khóa của lược đồ bằng cách gọi phương thức findAllKeys và ghi danh sách các khóa tìm được vào file đầu ra outputFile.

**Mục đích**

- Ghi danh sách tất cả các khóa tìm được vào một file dưới dạng văn bản.

**Cấu trúc và quy trình xử lý:**

1. **Tìm tất cả khóa**

        Set<Set<String>> allKeys = findAllKeys();

- Sử dụng findAllKeys() để lấy danh sách khóa.

- Kết quả được lưu trong Set<Set<String>> allKeys.

1. **Ghi vào file**

        try (BufferedWriter bw = new BufferedWriter(new FileWriter(outputFile))) {

            bw.write("Tổng số khóa tìm được: " + allKeys.size());

            bw.newLine();

            for (Set<String> key : allKeys) {

                bw.write(key.toString());

                bw.newLine();

            }

        }

- Sử dụng BufferedWriter để ghi:

+ Tổng số khóa: "Tổng số khóa tìm được: " + allKeys.size().

+ Từng khóa (dạng chuỗi Set.toString()).

### Tóm tắt các bước chương trình thực hiện:

1. **Đọc tệp đầu vào**

- Phương thức: readInput

- Mục đích:

+ Đọc file input.txt chứa tập thuộc tính và tập phụ thuộc hàm.

+ Lưu tập thuộc tính vào danh sách attributes.

+ Lưu tập phụ thuộc hàm vào danh sách dependencies.

Kết quả: Lưu trữ thông tin lược đồ vào hai cấu trúc dữ liệu:

+ attributes: Danh sách thuộc tính.

+ dependencies: Danh sách các phụ thuộc hàm.

1. **Phân tích chuỗi thuộc tính**

- Phương thức: parseAttributes

- Mục đích:

+ Loại bỏ ký tự không cần thiết (R()).

+ Tách danh sách thuộc tính từ dòng đầu vào bằng dấu phẩy.

- Kết quả: Lưu danh sách thuộc tính vào attributes.

1. **Phân tích phụ thuộc hàm**

- Phương thức: parseDependency

- Mục đích:

+ Phân tích chuỗi phụ thuộc hàm theo định dạng X -> Y.

+ Tách tập thuộc tính bên trái (left) và bên phải (right).

- Kết quả: Lưu từng phụ thuộc hàm vào danh sách dependencies dưới dạng đối tượng FunctionalDependency.

1. **Tính bao đóng của tập thuộc tính**

- Phương thức: closure

- Mục đích:

+ Tính bao đóng của một tập thuộc tính đầu vào (initialSet).

+ Xác định tất cả các thuộc tính có thể suy diễn được từ tập đó.

- Kết quả: Trả về tập bao đóng chứa tất cả các thuộc tính suy diễn được.

1. **Tìm tất cả các khóa**

- Phương thức: findAllKeys

- Mục đích:

+ Sử dụng bitmask để liệt kê tất cả các tập con của tập thuộc tính.

+ Kiểm tra bao đóng của từng tập con:

+ Nếu bao đóng chứa toàn bộ thuộc tính, tập con đó là một khóa.

- Kết quả: Trả về danh sách tất cả các tập thuộc tính thỏa mãn điều kiện là khóa.

1. **Ghi kết quả ra tệp**

- Phương thức: writeAllKeys

- Mục đích:

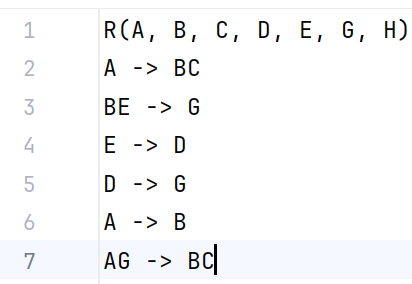
+ Ghi tổng số khóa và danh sách từng khóa vào file output.txt.

- Kết quả: File output.txt chứa:

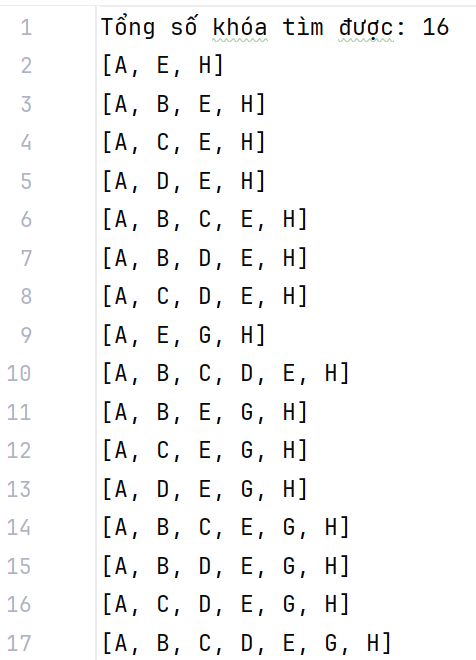
+ Tổng số khóa.

+ Danh sách chi tiết từng khóa.

### Hình ảnh các testcase:



Hình 32: Test case input cho tìm khóa của lược đồ CSDL



Hình 33: Test case output cho tìm khóa của lược đồ CSDL