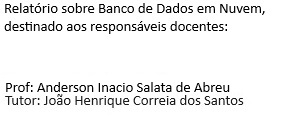
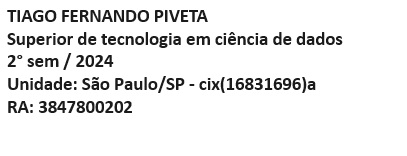
**Relatório de Aula Pratica- Arquitetura de Dados**

**Normalização de Tabela – Formas Normas**

****

****

Sumário

[1. O Que são Formas Normais e suas principais regras de normalização de tabelas: 2](#_Toc179310337)

[1.1. Introdução: 2](#_Toc179310338)

[1.2. Primeira Forma Normal (1FN): 2](#_Toc179310339)

[1.3. Segunda Forma Normal (2FN): 3](#_Toc179310340)

[1.4. Terceira Forma Normal (3FN): 3](#_Toc179310341)

[2. Criando tabelas 2FN a partir da tabela Dinner\_Service “1FN”: 3](#_Toc179310342)

[3. Transformando para 3FN: 12](#_Toc179310343)

[4. Visão Geral do DER: 13](#_Toc179310344)

[5. Salvando Arquivo .mwb: 15](#_Toc179310345)

[6. Criando um Database e Criando as tabelas: 18](#_Toc179310346)

[7. Inserindo Dados nas tabelas: 20](#_Toc179310347)

[8. Executando consultas SQL para validarmos as Inserções: 20](#_Toc179310348)

[9. Conclusão: 22](#_Toc179310349)

# O Que são Formas Normais e suas principais regras de normalização de tabelas:

## Introdução:

A normalização de dados é um processo essencial no design de bancos de dados relacionais. Ela visa organizar os dados para minimizar redundâncias e dependências, garantindo a integridade e a eficiência do banco de dados. As formas normais são etapas sequenciais desse processo, cada uma com critérios específicos que devem ser atendidos.

## Primeira Forma Normal (1FN):

**Definição**: Uma tabela está na Primeira Forma Normal (1FN) quando todos os seus atributos contêm valores atômicos e não há grupos repetidos.

**Características**:

* **Valores Atômicos**: Cada coluna deve conter apenas valores indivisíveis. Por exemplo, uma coluna de nomes não deve conter múltiplos nomes em uma única célula.
* **Ausência de Grupos Repetidos**: Não deve haver colunas que contenham listas ou conjuntos de valores. Cada entrada na tabela deve ser única e não deve haver repetição de grupos de dados.

## Segunda Forma Normal (2FN):

**Definição**: Uma tabela está na Segunda Forma Normal (2FN) quando está na 1FN e todos os atributos não-chave são totalmente dependentes da chave primária.

**Características**:

* **Dependência Total da Chave**: Cada atributo não-chave deve depender completamente da chave primária. Isso elimina dependências parciais, onde um atributo depende apenas de parte da chave primária em tabelas com chaves compostas.

## Terceira Forma Normal (3FN):

**Definição**: Uma tabela está na Terceira Forma Normal (3FN) quando está na 2FN e todos os atributos não-chave são mutuamente independentes, ou seja, não há dependências transitivas.

**Características**:

* **Eliminação de Dependências Transitivas**: Um atributo não-chave não deve depender de outro atributo não-chave. Cada atributo não-chave deve depender apenas da chave primária.

# Criando tabelas 2FN a partir da tabela Dinner\_Service “1FN”:

Nesta primeira etapa, vamos criar a tabela *dinner\_service* de acordo com a amostra fornecida e definir os tipos de dados apropriados para cada coluna. Esta tabela já está na Primeira Forma Normal (1FN), que garante que todos os dados sejam atômicos, ou seja, não contém valores repetidos ou campos com listas.

• **Requisitos para a Primeira Forma Normal (1FN):**

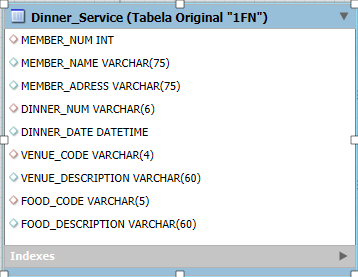
Para uma tabela estar na 1FN, ela deve atender a dois critérios principais:

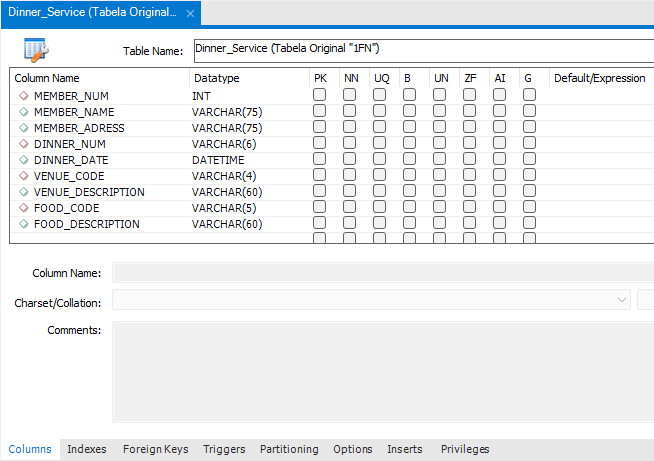
Valores Atômicos: Todos os valores nas colunas devem ser indivisíveis. Isso significa que não pode haver colunas com múltiplos valores (como listas) em uma única célula.

Ausência de Grupos Repetidos: Não deve haver repetição de grupos de dados. Cada campo deve armazenar apenas uma informação e não deve existir duplicidade em conjuntos de dados.

Com base nos dados apresentados, a tabela *dinner\_service* já está em conformidade com a 1FN, pois não apresenta valores repetidos nem campos compostos por mais de um valor.

**2.1- Criando a Tabela Dinner\_Service**:





**2.1.2- Análise da tabela 1FN: A tabela Dinner\_Service na 1FN contém os seguintes atributos**:

* MEMBER\_NUM (INT) *“Número de identificação do membro (inteiro)”*
* MEMBER\_NAME (VARCHAR(75)) *“Nome do membro (cadeia de texto, até 75 caracteres)”*
* MEMBER\_ADRESS (VARCHAR(75)) *“Endereço do membro (cadeia de texto, até 75 caracteres”*
* DINNER\_NUM (VARCHAR(6)) *”Número do jantar (cadeia de texto, até 6 caracteres)”*
* DINNER\_DATE (DATETIME) *“Data e hora do jantar (formato de data e hora)”*
* VENUE\_CODE (VARCHAR(4)) *“Código do local (cadeia de texto, até 4 caracteres)”*
* VENUE\_DESCRIPTION (VARCHAR(60)) *“Descrição do local (cadeia de texto, até 60 caracteres)”*
* FOOD\_CODE (VARCHAR(5)) *“Código do alimento (cadeia de texto, até 5 caracteres)”*
* FOOD\_DESCRIPTION (VARCHAR(60)) *“Descrição do alimento (cadeia de texto, até 60 caracteres)”*

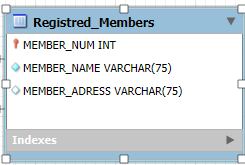
***2.1.3-* Explicação dos Tipos de Dados:**

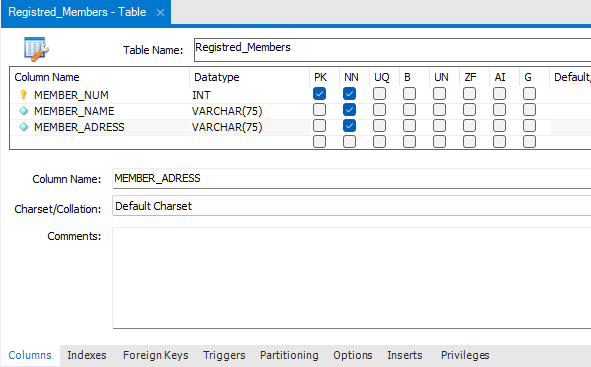
* **MEMBER\_NUM (INT):** Usamos o tipo INT para armazenar o número de identificação do membro, pois se trata de um número inteiro.
* **MEMBER\_NAME (VARCHAR(75)):** O tipo VARCHAR(75) é utilizado para armazenar o nome do membro, pois é um campo de texto que pode variar em tamanho, até 75 caracteres.
* **MEMBER\_ADDRESS (VARCHAR(75)):** Utilizamos VARCHAR(75) para o endereço do membro, já que pode ser uma cadeia de texto variável de até 75 caracteres.
* **DINNER\_NUM (VARCHAR(6)):** O número do jantar é armazenado como VARCHAR(6), pois é uma combinação de números ou letras de até 6 caracteres.
* **DINNER\_DATE (DATETIME):** Para a data e hora do jantar, o tipo DATETIME é adequado, pois permite registrar tanto a data quanto a hora do evento.
* **VENUE\_CODE (VARCHAR(4)):** O código do local é armazenado como VARCHAR(4), sendo uma cadeia de texto curta que representa o local do jantar.
* **VENUE\_DESCRIPTION (VARCHAR(60)):** A descrição do local é armazenada como VARCHAR(60), uma cadeia de texto de até 60 caracteres.
* **FOOD\_CODE (VARCHAR(5)):** O código do alimento, por ser uma sequência curta, é armazenado como VARCHAR(5).
* **FOOD\_DESCRIPTION (VARCHAR(60)):** A descrição do alimento também é armazenada como VARCHAR(60).

* **Divisão da Tabela dinner\_service em Tabelas para Atingir a 2FN:**

A tabela dinner\_service foi dividida nas seguintes tabelas para garantir a conformidade com a 2FN. O critério de divisão foi separar os diferentes assuntos (dados de membros, informações de jantares, locais e alimentos) em entidades distintas.  
  
  
**2.1.4- Criando a Tabela Registred\_Members**:

Essa tabela contém informações sobre os membros registrados no sistema. A separação dos dados de membros segue o princípio da 2FN, pois os dados de membros dependem exclusivamente de *MEMBER\_NUM*, que se tornou a chave primária.

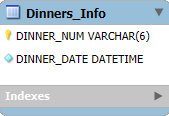
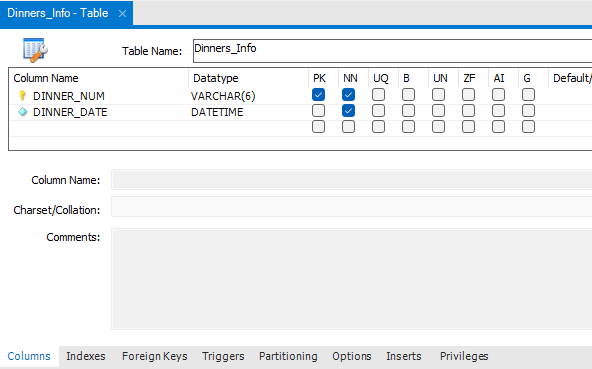




**2.1.5 - Análise da Tabela Registred\_Members que contém os seguintes atributos**:

* MEMBER\_NUM (INT) - Chave primária ‘Número de identificação do membro (chave primária)’
* MEMBER\_NAME (VARCHAR(75)) ‘Nome do membro’
* MEMBER\_ADRESS (VARCHAR(75)) ‘Endereço do membro’
* **Chave Primária:** *MEMBER\_NUM* é a chave primária desta tabela, garantindo que cada membro tenha um identificador único.
* **Atributos Dependentes:** Todos os outros atributos, como *MEMBER\_NAME* e *MEMBER\_ADDRESS*, dependem totalmente do *MEMBER\_NUM*, satisfazendo a 2FN.

**2.1.6- Criando a Tabela Dinners\_Info:**

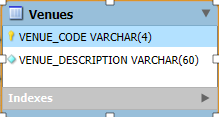
 Esta tabela armazena informações sobre cada jantar registrado no sistema, incluindo o número do jantar e a data. A separação é feita para garantir que os dados de jantares não fiquem misturados com dados de membros ou outros tópicos.

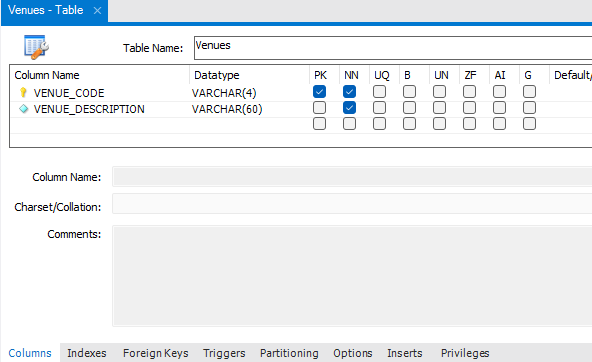
**2.1.7- Análise da Tabela Dinners\_Info que contém os seguintes atributos:**

* DINNER\_NUM (VARCHAR(6)) - Chave primária ‘Número do jantar (chave primária)’
* DINNER\_DATE (DATETIME) ‘Data e hora do jantar’
* **Chave Primária**: *DINNER\_NUM* é a chave primária, garantindo que cada jantar tenha um identificador único.
* **Atributos Depende***ntes*: O atributo *DINNER\_DATE* depende exclusivamente de *DINNER\_NUM*, o que elimina qualquer dependência parcial.

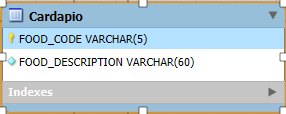
**2.1.8- Criando a Tabela Venues:**

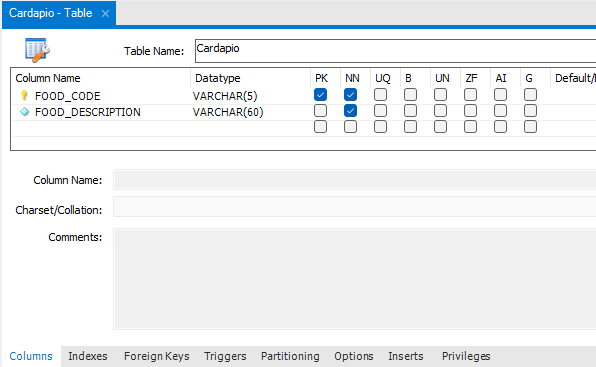
A tabela Venues contém informações sobre os locais onde os jantares ocorrem. A separação dos dados de locais ajuda a eliminar redundâncias e dependências parciais.





* + 1. **Análise da Tabela Venues que contém os seguintes atributos:**
* VENUE\_CODE (VARCHAR(4)) - Chave primária ‘Código do local (chave primária)’
* VENUE\_DESCRIPTION (VARCHAR(60)) ‘Descrição do local’
* **Chave Primária**: *VENUE\_CODE* é a chave primária desta tabela.
* **Atributos Dependentes**: A descrição do local (*VENUE\_DESCRIPTION*) depende exclusivamente do *VENUE\_CODE*.

**2.1.10- Criando a Tabela Cardapio:**



## 

## 

**2.1.11- Análise da Tabela Cardapio que contém os seguintes atributos:**

Essa tabela contém informações sobre os alimentos servidos durante os jantares. A separação desses dados ajuda a evitar redundâncias e garantir que os alimentos estejam associados corretamente aos jantares por meio de chaves estrangeiras.

* FOOD\_CODE (VARCHAR(5)) - Chave primária ‘Código do alimento (chave primária)’
* FOOD\_DESCRIPTION (VARCHAR(60)) ‘Descrição do alimento’
* **Chave Primária**: *FOOD\_CODE* é a chave primária desta tabela.
* **Atributos Dependentes**: A descrição do alimento (*FOOD\_DESCRIPTION*) depende exclusivamente de *FOOD\_CODE*.

**2.1.12- Definindo as Chaves estrangeiras na Tabela Dinner\_Service**:

* *TABELA DINNER\_SERVICE COM REGISTRED\_MEMBERS*:

**Foreign Key Name**: fk\_dinnerservice\_registredmembers

**Referenced Table**: 'mydb' . 'Registred\_Members'

**Column**: MEMBER\_NUM

**Referenced Column**: MEMBER\_NUM

* *TABELA DINNER\_SERVICE COM DINNER\_INFO*:

**Foreign Key Name**: fk\_dinnerservice\_dinners

**Referenced Table**: 'mydb' . 'Dinners\_Info'

**Column**: DINNER\_INFO

**Referenced Column**: DINNER\_INFO

* *TABELA DINNER\_SERVICE COM VENUES*:

**Foreign Key Name**: fk\_dinnerservice\_venues

**Referenced Table**: 'mydb' . 'Venues'

**Column**: VENUE\_CODE

**Referenced Column**: VENUE\_CODE

* *TABELA DINNER\_SERVICE COM CARDAPIO*:

**Foreign Key Name**: fk\_dinnerservice\_cardapio

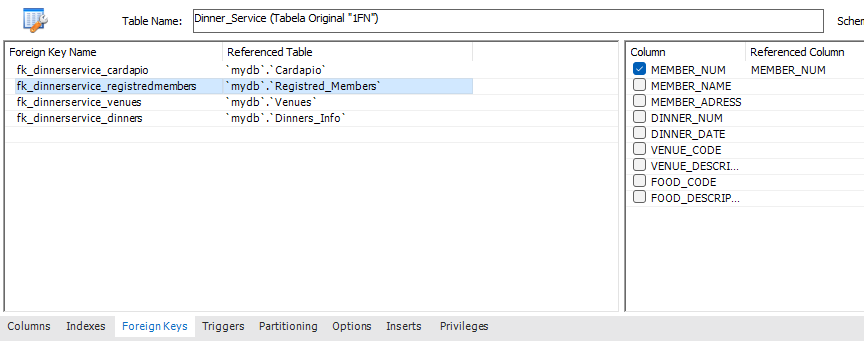
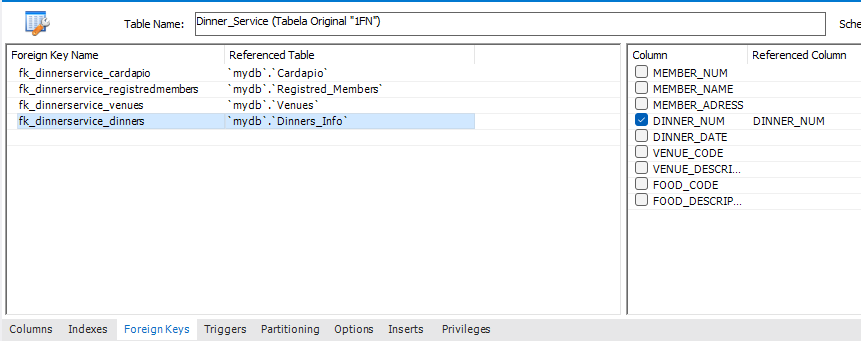
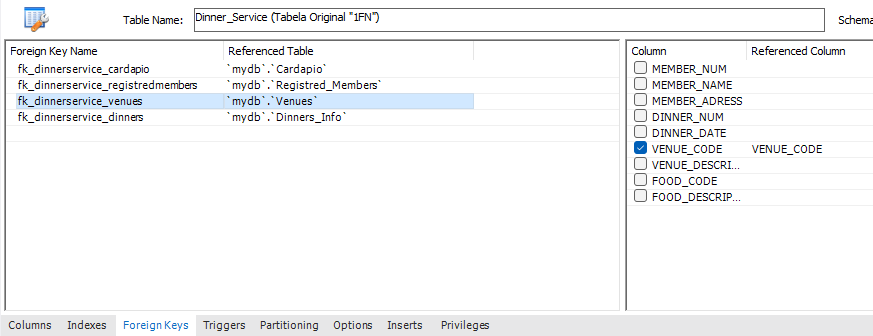
**Referenced Table**: 'mydb' . 'Cardapio'

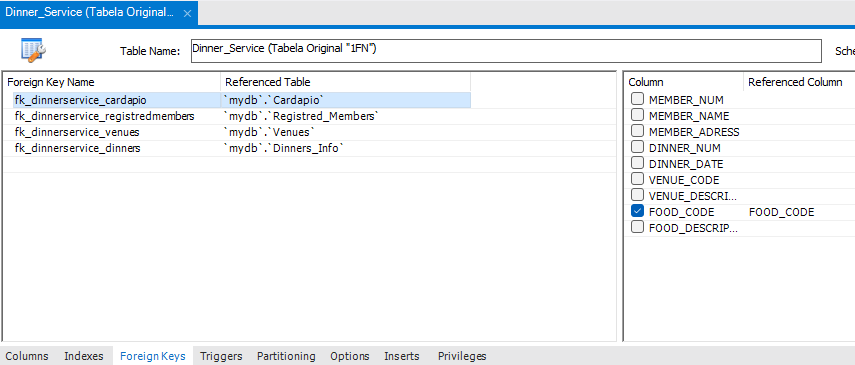
**Column**: FOOD\_CODE

**Referenced Column**: FOOD\_CODE

**2.1.13- Relacionamentos Criados na 2FN:**

Os relacionamentos criados entre as tabelas foram configurados com base nas chaves primárias e estrangeiras:

* **Relacionamento *Dinner\_Service* e *Registred\_Members*:** Um membro pode participar de múltiplos jantares, mas cada jantar está associado a um único membro. Este é um relacionamento **um-para-muitos**.
* **Relacionamento *Dinner\_Service* e *Dinners\_Info*:** Um jantar pode ser relacionado a vários registros de membros, mas cada entrada de *Dinner\_Service* faz referência a um único jantar. Relacionamento ***um-para-muitos***.
* **Relacionamento *Dinner\_Service* e *Venues*:** Vários jantares podem ocorrer no mesmo local, mas cada registro de *Dinner\_Service* está associado a um único local. Relacionamento ***um-para-muitos.***
* **Relacionamento *Dinner\_Service* e *Cardapio*:** Vários jantares podem ter o mesmo item do cardápio, mas cada registro de ***Dinner\_Service*** está associado a um único item do cardápio. Relacionamento ***um-para-muitos***.

  
  
Após a normalização para a Segunda Forma Normal (2FN), as tabelas estão organizadas de forma que todas as dependências parciais foram eliminadas. Cada tabela agora contém apenas atributos que dependem exclusivamente de sua chave primária, e as redundâncias foram removidas. A tabela ***Dinner\_Service*** serve como a relação entre membros, jantares, locais e alimentos, garantindo que os dados estejam estruturados corretamente.

# Transformando para 3FN:

O objetivo principal desta etapa é ajustar o banco de dados para que ele atenda às regras da 3FN. Isso significa garantir que cada atributo não-chave dependa diretamente da chave primária e que não existam dependências transitivas (ou seja, não deve haver uma relação indireta entre os atributos).

* Explicações e Transformações Realizadas:
  1. **Tabela Registred\_Members e Members\_Adress:**
* **Motivo da Criação da Tabela Members\_Adress:**  
  A tabela original *Registred\_Members* continha dados repetidos de endereço para o mesmo cliente, o que gerava duplicação de dados e problemas de inserção. Para resolver isso, criamos a tabela ***Members\_Adress***, que separa os dados de endereço dos dados de identificação dos membros. Com essa mudança:
* A tabela ***Registred\_Members*** passou a conter apenas as colunas ***MEMBER\_NUM*** e ***MEMBER\_NAME***, ficando com os dados essenciais de identificação.
* A tabela ***Members\_Adress*** foi criada com as colunas ***MEMBER\_NUM***, ***MEMBER\_ADRESS\_1***, e ***MEMBER\_ADRESS\_2***, para armazenar múltiplos endereços de membros.
* **Relacionamento:**  
  Criamos uma chave estrangeira ligando ***Registred\_Members*** a ***Members\_Adress***, através da coluna ***MEMBER\_NUM***. O relacionamento é ***um-para-muitos***, visto que um cliente pode ter mais de um endereço.

**3.2- Tabela Dinner\_Requests**:

A tabela *Dinner\_Requests* foi criada para organizar as solicitações de jantares (dinners) feitas pelos membros. Essa tabela contém as seguintes colunas:

* **Requests\_id**: Chave primária.
* **MEMBER\_NUM**: Referência ao membro que fez a solicitação (chave estrangeira ligada a *Registred\_Members*).
* **DINNER\_NUM**: Referência ao jantar específico (chave estrangeira ligada a *Dinners\_Info*).
* **FOOD\_CODE**: Referência ao cardápio (chave estrangeira ligada a *Cardapio*).
* **Relacionamento:**  
  A tabela Dinner\_Requests recebe as seguintes chaves estrangeiras:
* Ligação com *Dinner\_Service* através de *MEMBER\_NUM* (um membro pode fazer muitos pedidos).
* Ligação com *Cardapio* através de *FOOD\_CODE* (um prato pode ser pedido em muitos jantares).
* Ligação com *Dinners\_Info* através de *DINNER\_NUM* (um jantar pode ter múltiplas solicitações de diferentes membros).

**3.3- Tabela Restructured\_Dinner\_Service:**

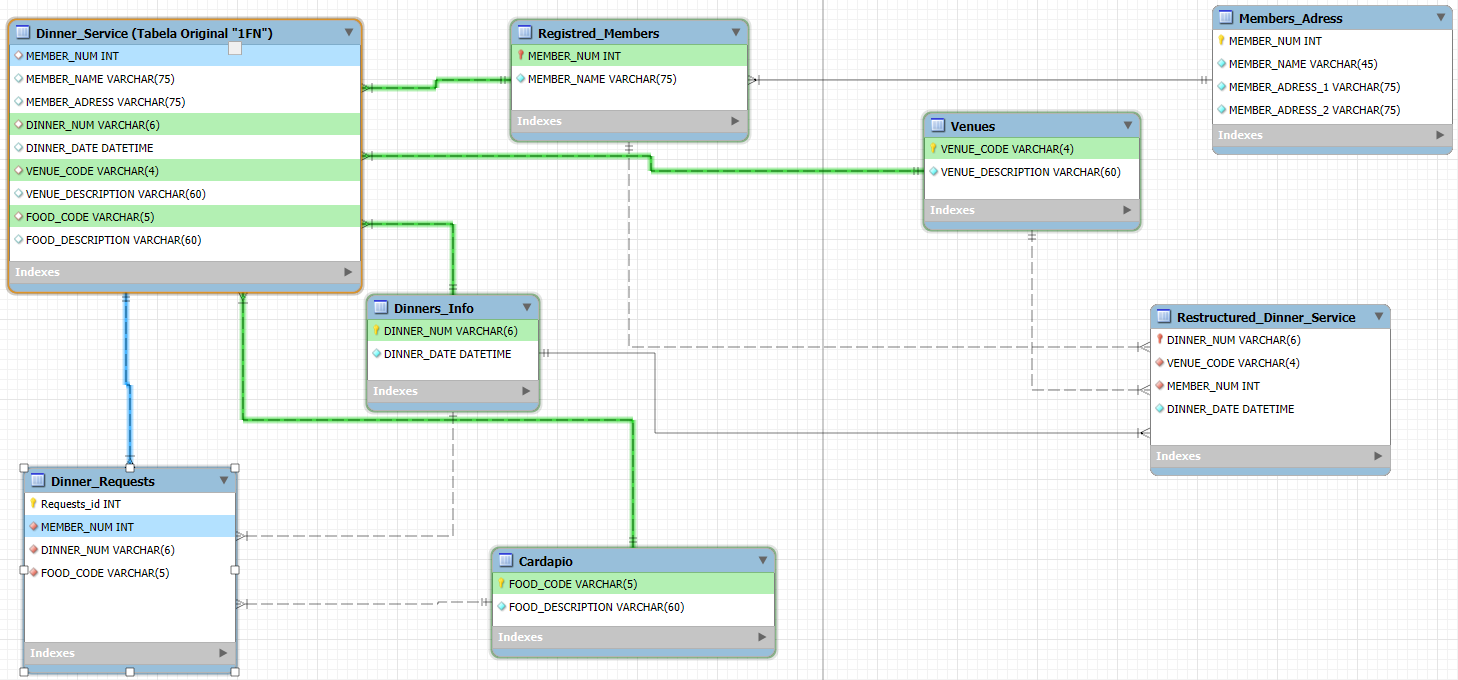
A tabela ***Restructured\_Dinner\_Service*** foi criada para representar de forma mais clara os jantares realizados, vinculando diretamente os jantares, os membros participantes e os locais onde os jantares ocorreram. Ela contém as colunas:

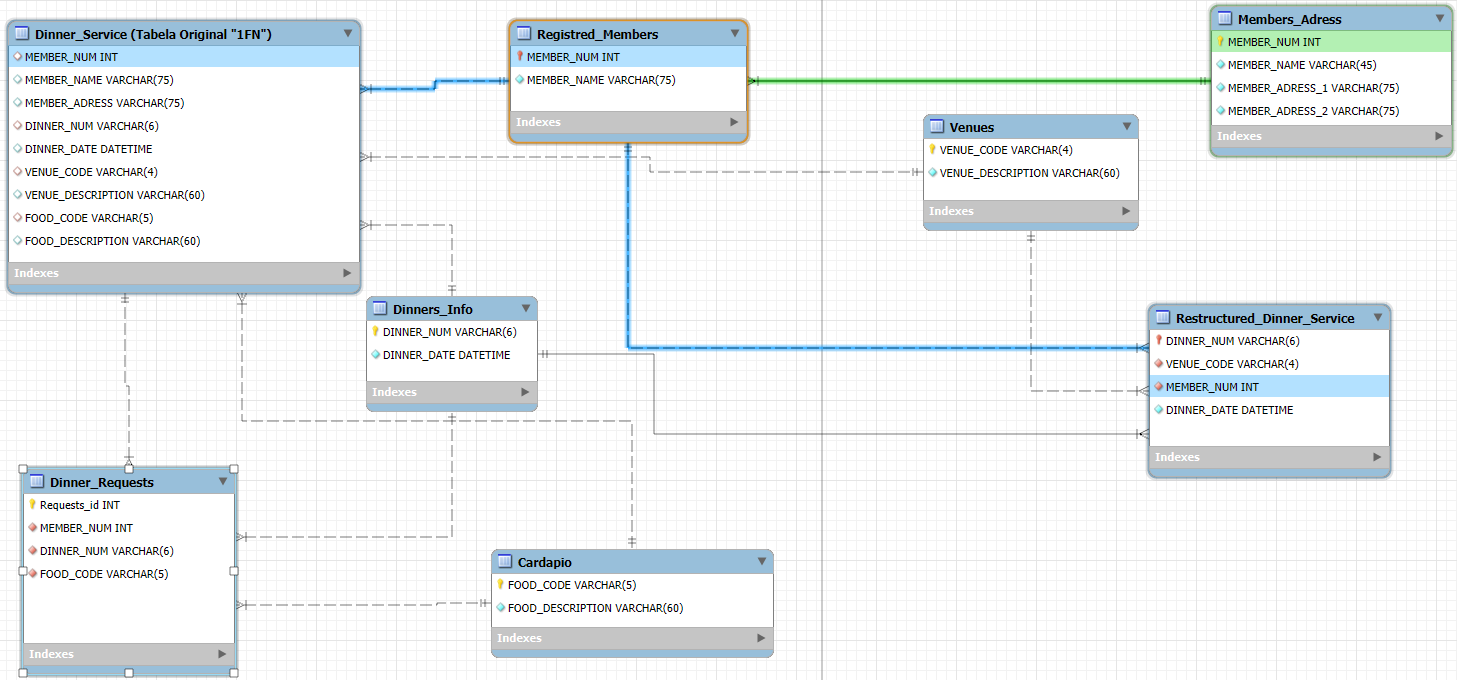
* **DINNER\_NUM**: Chave primária, representando o número do jantar.
* **VENUE\_CODE**: Referência ao local do jantar (chave estrangeira ligada a *Venues*).
* **MEMBER\_NUM**: Referência ao membro (chave estrangeira ligada a *Registred\_Members*).
* **DINNER\_DATE**: Data do jantar.
* **Relacionamento:**

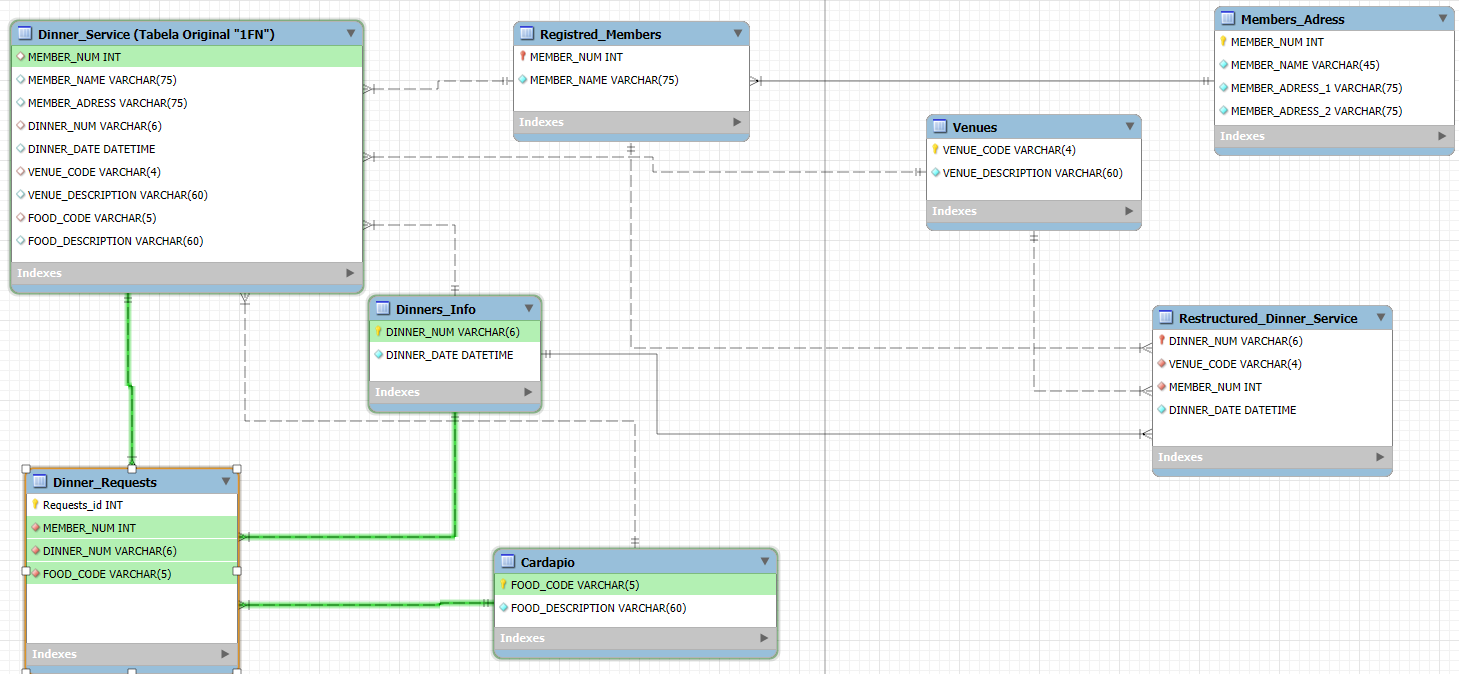
A tabela *Restructured\_Dinner\_Service* recebe chaves estrangeiras:

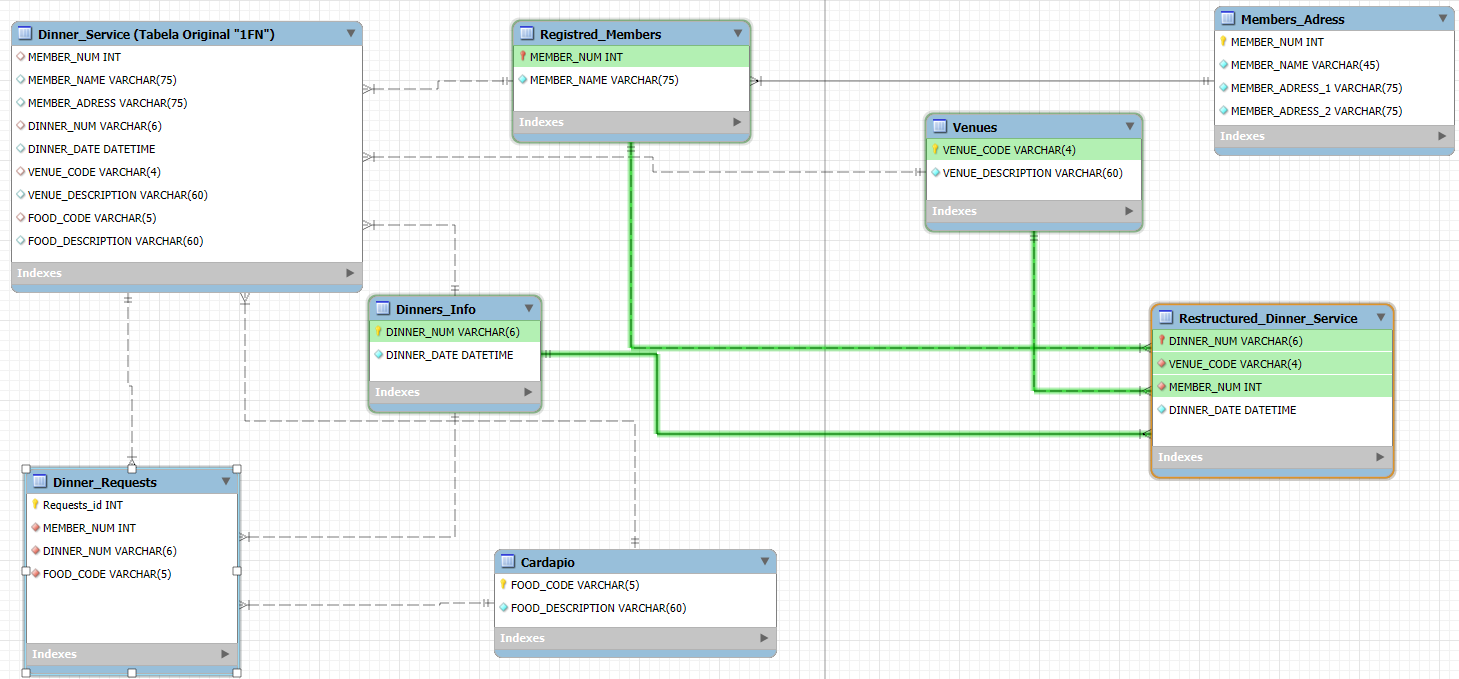
* Ligação com *Dinner\_Service* via *MEMBER\_NUM*.
* Ligação com *Cardapio* via *FOOD\_CODE* (definindo quais pratos foram servidos).
* Ligação com *Dinners\_Info* via *DINNER\_NUM*.
  1. **Breve Discussão sobre a Transformação para 3FN:**
* **Dependências Transitivas Removidas:**  
  O principal problema que foi resolvido ao passar para a 3FN foi a eliminação das **dependências transitivas**. Por exemplo, os dados de endereço de membros estavam sendo duplicados junto com as informações dos membros, o que criou a necessidade de uma nova tabela para endereços (Members\_Adress). Isso garantiu que os dados estivessem devidamente normalizados, sem redundância.
* **Integridade dos Dados:**  
  Agora, todos os dados estão organizados de maneira que cada atributo dependa **diretamente da chave primária** de sua tabela. Isso significa que:
* Não há dependências indiretas ou transitivas.
* Não há duplicação desnecessária de dados.
* Inserções e atualizações podem ser feitas com mais facilidade, sem risco de inconsistências.
* **Chaves Estrangeiras e Relacionamentos:**  
  Todos os relacionamentos entre tabelas foram definidos através de **chaves estrangeiras**, garantindo a integridade referencial. Isso também ajuda na criação de **consultas mais eficientes** e melhora a estrutura do banco de dados para operações futuras.

# Visão Geral do DER:



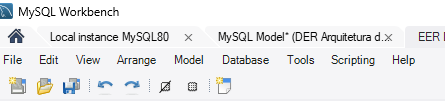




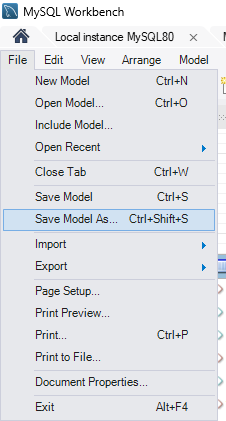


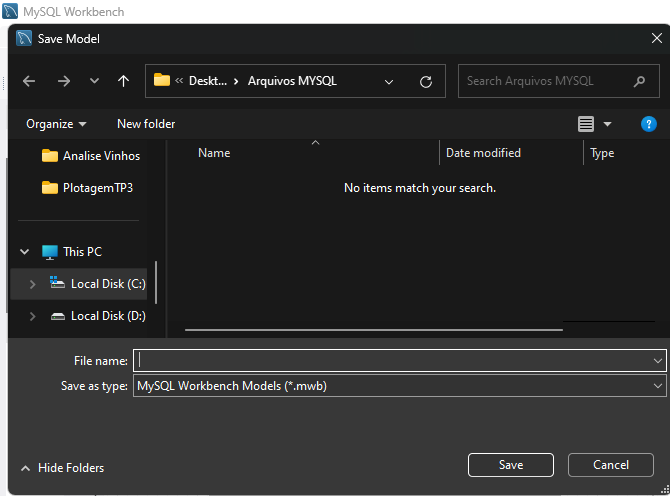
# Salvando Arquivo .mwb:

* **Clicar em File**



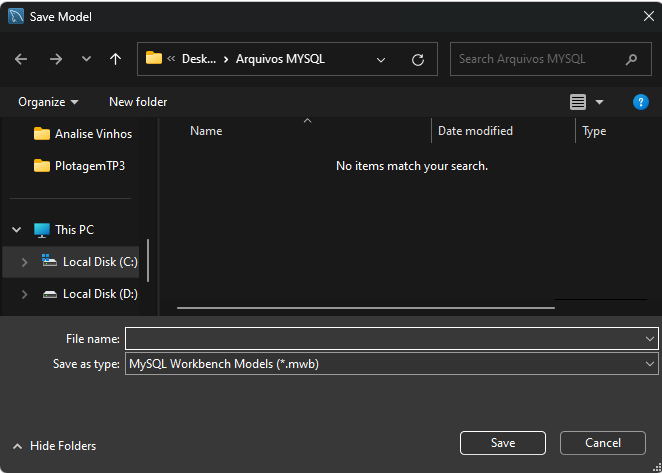
* Clicar em Save Model As:



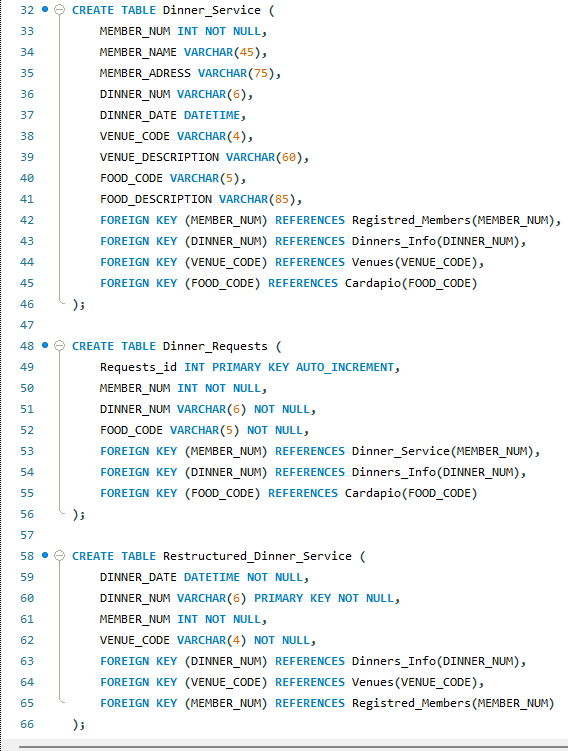
* **Escolher Local onde irá salvar o arquivo e inserir um nome para o arquivo:**

***Insira um nome para o arquivo.***

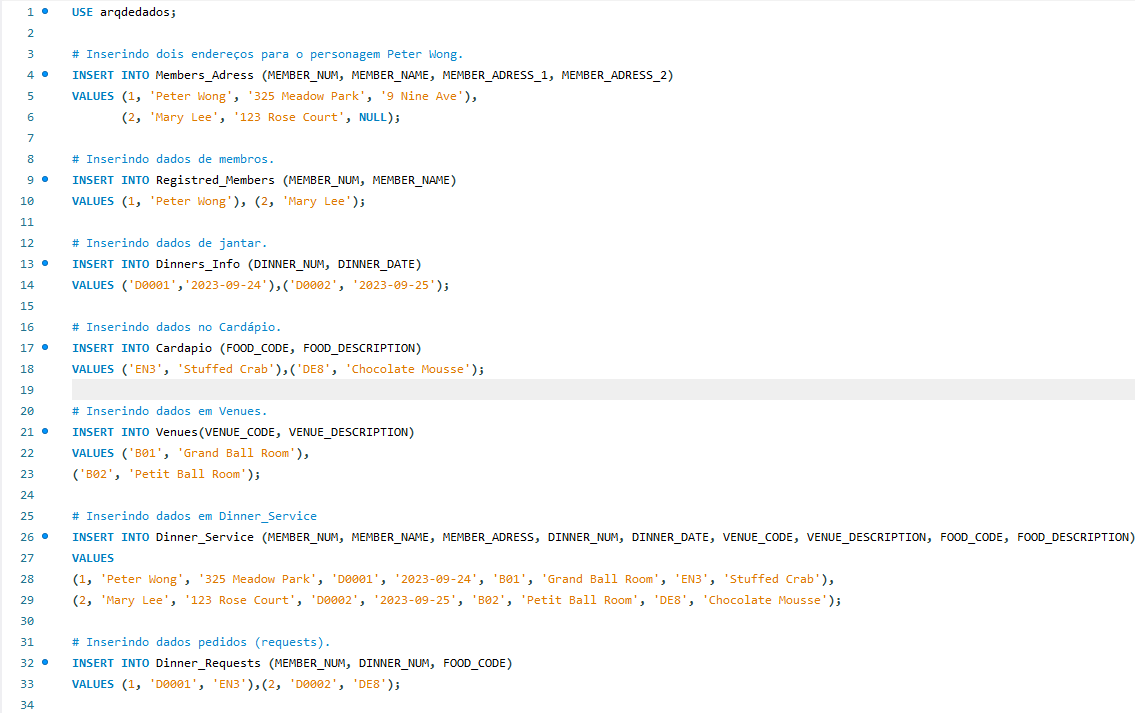
* Clicar em Save:



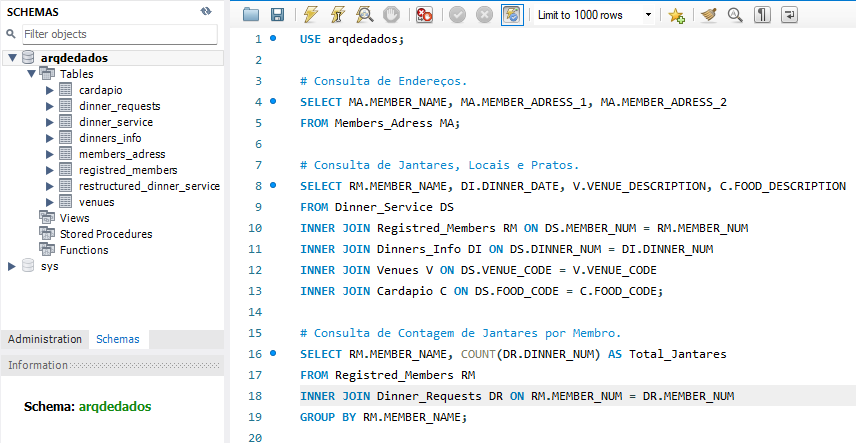
# Criando um Database e Criando as tabelas:

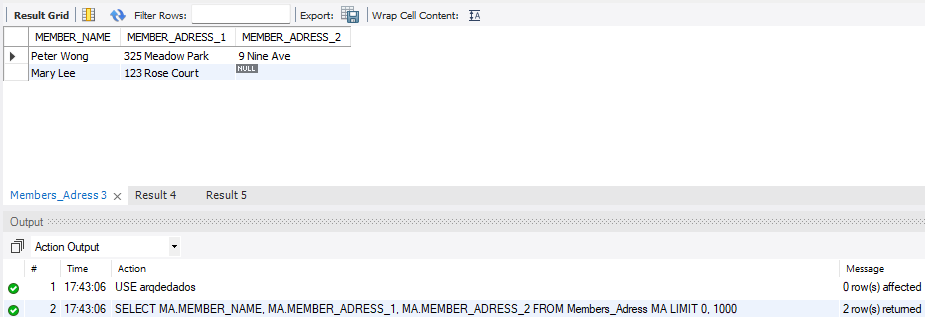


# Inserindo Dados nas tabelas:



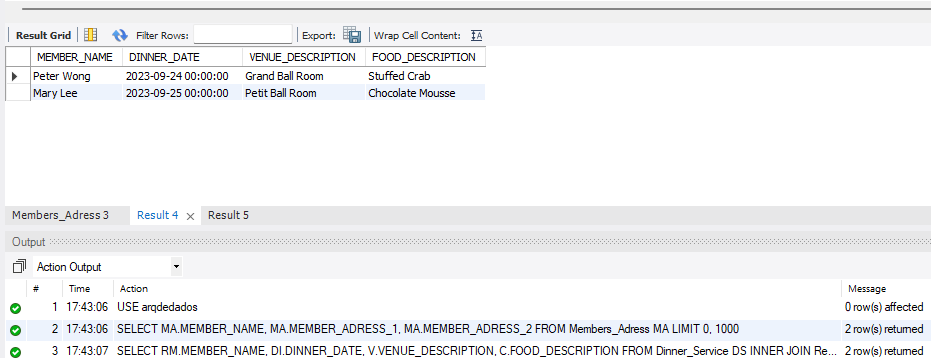
# Executando consultas SQL para validarmos as Inserções:



**8.1-Resultado da primeira Consulta de Endereços:**

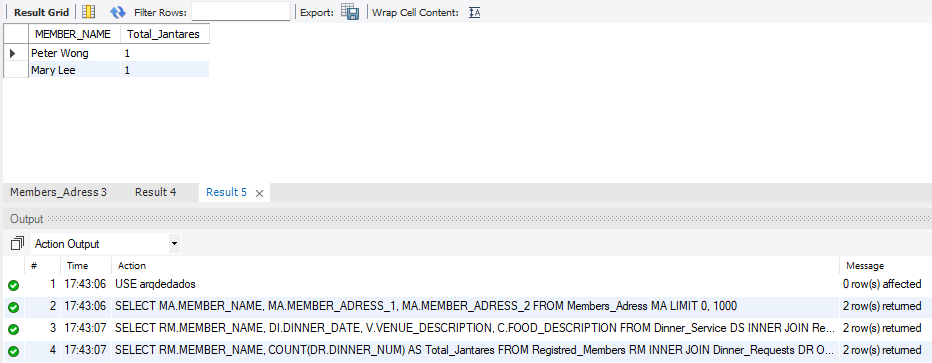
Isso mostra que o membro Peter Wong tem dois endereços, enquanto Mary Lee tem apenas um, o que corresponde à estrutura da tabela e ao objetivo de separar os endereços para evitar duplicações de membros.

**8.2-Resultado da Segunda Consulta de Jantares, Locais e Pratos:**



Essa consulta mostra a associação correta entre os membros, o local do jantar, o prato servido e a data do jantar. Peter Wong participou de um jantar no "Grand Ball Room" com o prato "Stuffed Crab", e Mary Lee esteve no "Petit Ball Room" com "Chocolate Mousse".

**8.3-Resultado da Terceira Consulta Contagem de Jantares por Membro:**



Ambos os membros, Peter Wong e Mary Lee, participaram de um jantar cada, o que é consistente com os dados que você inseriu nas tabelas.

# Conclusão:

O projeto desenvolvido ao longo desta atividade seguiu todas as etapas necessárias para criar um banco de dados eficiente, desde a modelagem inicial até a aplicação de técnicas de normalização e o desenvolvimento de consultas SQL. O objetivo foi garantir que o banco de dados fosse estruturado de maneira a otimizar o armazenamento, manter a integridade dos dados e permitir a extração de informações de forma precisa e eficiente.

**1. Modelagem do Banco de Dados**:

O processo começou com a modelagem do banco de dados, representando as entidades, atributos e relacionamentos necessários para a estrutura do sistema. O Diagrama Entidade-Relacionamento (DER) foi desenvolvido para representar graficamente como os dados se relacionam, destacando elementos como membros registrados, cardápios de jantar, locais de eventos, pedidos de jantar e as informações dos eventos. Essa fase de planejamento foi crucial para evitar problemas futuros, como redundância de dados e dificuldades de manutenção.

2. Aplicação da Normalização:

A normalização foi aplicada de forma gradual até atingir a terceira forma normal (3FN). Esse processo envolveu a reestruturação de tabelas para eliminar redundâncias e dependências parciais ou transitivas. Um exemplo claro foi a criação da tabela *Members\_Adress*, que permitiu armazenar múltiplos endereços para um mesmo membro, algo que anteriormente estava gerando duplicação de dados no sistema.

A normalização trouxe os seguintes benefícios:

**Redução da Redundância de Dados**: Eliminamos dados repetidos, o que economiza espaço e evita discrepâncias. Agora, cada dado é armazenado uma única vez, e eventuais atualizações são feitas de forma centralizada.

**Melhoria na Consistência**: Garantimos que as informações sejam sempre consistentes em todas as consultas e operações do banco, já que não há mais cópias desnecessárias de dados em diferentes tabelas.

**Facilidade de Expansão**: A estrutura normalizada possibilita a inclusão de novas tabelas ou atributos de forma organizada, sem a necessidade de grandes reformulações no modelo existente.

**3. Criação das Tabelas e Definição de Chaves:**

Após o processo de normalização, foi necessário criar as tabelas no banco de dados, utilizando comandos SQL. As tabelas, como *Registred\_Members*, *Dinner\_Service*, *Dinner\_Requests, Cardapio* e *Venues*, foram criadas de forma a garantir que cada uma tivesse uma chave primária bem definida. Além disso, as chaves estrangeiras foram configuradas adequadamente para garantir a integridade referencial, assegurando que os dados entre as tabelas se relacionem de maneira lógica e consistente.

Esse relacionamento entre as tabelas foi essencial para construir um sistema que pudesse lidar com operações complexas e inter-relacionadas, como consultar membros e seus respectivos jantares, locais de eventos, e itens de cardápio.

**4. Inserção e Manipulação de Dados:**

Com as tabelas criadas e estruturadas corretamente, partimos para a inserção de dados relevantes no banco. Os dados de membros, cardápios, locais e eventos foram inseridos nas tabelas através de comandos SQL, o que permitiu testar o funcionamento do sistema na prática. Essa etapa foi fundamental para verificar a integridade do banco de dados e identificar possíveis problemas de consistência.

**5. Consultas SQL e Análises:**

Para validar o modelo criado, realizamos diversas consultas SQL que demonstraram a eficiência do banco de dados normalizado. As consultas foram capazes de:

Exibir os nomes e endereços dos membros registrados.

Listar as informações sobre os jantares, incluindo datas, locais e cardápios.

Contabilizar o número de jantares associados a cada membro.

Essas consultas confirmaram que o sistema estava funcionando conforme o esperado, com os dados sendo extraídos de maneira precisa, sem duplicações ou inconsistências. Esse tipo de análise é crucial para qualquer sistema de banco de dados, pois permite obter informações estratégicas a partir dos dados armazenados.

**6. Importância da Normalização**:

A normalização do banco de dados demonstrou ser essencial para o sucesso do projeto. Além de eliminar redundâncias e melhorar a organização dos dados, ela também:

**Melhora a Performance:** Embora o processo de normalização envolva a criação de mais tabelas, ele otimiza as consultas e operações, pois evita o processamento de grandes quantidades de dados duplicados.

**Facilita a Manutenção:** Um banco de dados bem estruturado é mais fácil de manter e atualizar. Alterações em um registro são refletidas em todas as consultas automaticamente, sem a necessidade de alterações manuais em diferentes locais.

**Escalabilidade:** A estrutura normalizada prepara o banco de dados para futuras expansões. Caso novos módulos ou funcionalidades precisem ser adicionados ao sistema, a organização lógica existente facilita a integração.

**7. Exportação e Salvamento do Modelo:**

Por fim, o modelo de banco de dados foi salvo e exportado no ***formato .mwb***, permitindo o compartilhamento do diagrama e dos esquemas com outros membros da equipe ou futuros desenvolvedores. Isso garante que o modelo seja documentado e possa ser revisado e aprimorado conforme as necessidades do projeto evoluam.

*Considerações Finais:*

Este projeto ilustrou a importância de um banco de dados bem modelado e normalizado, não apenas para garantir a integridade dos dados, mas também para otimizar as operações e facilitar a manutenção. A partir de uma estrutura eficiente, é possível realizar consultas complexas de forma eficaz, atender a novas demandas e adaptar-se ao crescimento dos dados com facilidade. A normalização até a 3FN demonstrou ser uma prática essencial para eliminar redundâncias e promover um sistema escalável, confiável e de fácil administração.