Sumário

[Seção 1 7](#_Toc78994973)

[Seção 2: Um pouco da teoria sem esquecer da prática 7](#_Toc78994974)

[Arquitetura de Software 7](#_Toc78994975)

[Modelagem Lógica e Física 7](#_Toc78994976)

[Modelagem Física 8](#_Toc78994977)

[Comparando CHAR e VARCHAR 9](#_Toc78994978)

[Comparando ENUM e Numéricos 9](#_Toc78994979)

[Seção 3: Elevando o nível – Comandos que você não pode deixar de saber! 10](#_Toc78994980)

[Tipos Nulos e Inteiros 10](#_Toc78994981)

[Conhecendo as Projeções 10](#_Toc78994982)

[Seleções 10](#_Toc78994983)

[Seção 4: Exercícios de revisão 11](#_Toc78994984)

[Seção 5: Lógica de Predicados 13](#_Toc78994985)

[A Tabela Verdade 13](#_Toc78994986)

[Operadores Lógicos – Prática 14](#_Toc78994987)

[Performance com Operadores Lógicos 14](#_Toc78994988)

[Seção 6: Exercícios de Fixação 15](#_Toc78994989)

[Seção 7: Mais alguns comandos básicos 15](#_Toc78994990)

[Filtrando Valores Nulos 15](#_Toc78994991)

[A cláusula Update 15](#_Toc78994992)

[A cláusula DELETE 16](#_Toc78994993)

[Seção 8: Modelando Bancos de Dados para Sistemas 16](#_Toc78994994)

[Começando a Modelar 16](#_Toc78994995)

[A história da Modelagem 16](#_Toc78994996)

[Primeira Forma Normal 16](#_Toc78994997)

[Cardinalidade e Obrigatoriedade 17](#_Toc78994998)

[Modelo Lógico para Físico – Parte 01 19](#_Toc78994999)

[Modelo Físico – Parte 02 19](#_Toc78995000)

[Entendendo a Foreign Key – Parte 01 20](#_Toc78995001)

[Foreign Key – Parte 02 20](#_Toc78995002)

[Serviços do Banco, Password e Sublime Text 21](#_Toc78995003)

[Inserindo Dados 21](#_Toc78995004)

[Inserções em Relacionamentos 1 x 1 21](#_Toc78995005)

[Inserções 1 x N 22](#_Toc78995006)

[Seção 9: Juntando as peças 22](#_Toc78995007)

[Seleção e Projeção 22](#_Toc78995008)

[Junção 22](#_Toc78995009)

[Inner Join 23](#_Toc78995010)

[Comandos de DML 23](#_Toc78995011)

[DDL – Modificando Tabelas 24](#_Toc78995012)

[Seção 10: Exercícios de Fixação 25](#_Toc78995013)

[Seção 11: Aprofundando 25](#_Toc78995014)

[Funções – IFNULL 25](#_Toc78995015)

[Views 25](#_Toc78995016)

[Operações de DML em Views 26](#_Toc78995017)

[Ordenando Dados 26](#_Toc78995018)

[Seção 12: Outras Notações 27](#_Toc78995019)

[Notação de Cross e o StarUML 27](#_Toc78995020)

[Seção 13: Programando em MySQL 29](#_Toc78995021)

[Delimiter 29](#_Toc78995022)

[Iniciando com Programação em Bancos de Dados 29](#_Toc78995023)

[Procedures no Mundo Real 29](#_Toc78995024)

[Procedures com Query – Parâmetros 29](#_Toc78995025)

[Seção 14: Agregue, some, me dê a média e tudo mais – Funções de Agregação 29](#_Toc78995026)

[Group by, count, max, min, avg e funções do MySQL 29](#_Toc78995027)

[Tudo certo, mas agora eu quero a soma – Utilizando o SUM! 29](#_Toc78995028)

[Seção 15: Subconjunto do Conjunto 29](#_Toc78995029)

[Utilizando Subqueries 29](#_Toc78995030)

[Somei as colunas. Mas e as linhas? Operações aritméticas. 29](#_Toc78995031)

[Seção 16: Organizando a casa – O dicionário de dados! 29](#_Toc78995032)

[Verificando e alterando a estrutura de uma tabela, seus objetos e charsets 29](#_Toc78995033)

[Organizando as chaves e introdução às bases de dicionário 29](#_Toc78995034)

[Constraints Nomeadas x Dicionário de Dados 29](#_Toc78995035)

[Aprofundando com constraints – Querys de Dicionários 29](#_Toc78995036)

[Seção 17: Vamos Revisar? 29](#_Toc78995037)

[Revisão ilustrada sobre Foreign Keys 29](#_Toc78995038)

[Seção 18: Aprendeu? Saiu o sistema do Seu José 29](#_Toc78995039)

[Seção 19: Automatizando as coisas – Triggers 29](#_Toc78995040)

[Introdução às fantásticas Triggers 29](#_Toc78995041)

[Agora é pra valer! Trigger na prática 29](#_Toc78995042)

[Triggers para bancos de backups 29](#_Toc78995043)

[After, before, insert, delete ou update? Os eventos de uma trigger 29](#_Toc78995044)

[Quem mexeu no meu dado? Auditando uma tabela com trigger 29](#_Toc78995045)

[Seção 20: Mais modelagem 29](#_Toc78995046)

[Seção 21: Programe, programe! 30](#_Toc78995047)

[Introdução aos cursores 30](#_Toc78995048)

[E mais cursores! 30](#_Toc78995049)

[Seção 22: Normalizando mais! 30](#_Toc78995050)

[2 e 3 Formas Normais! 30](#_Toc78995051)

[Vamos praticar? Parte 1 30](#_Toc78995052)

[Criando as Constraints 30](#_Toc78995053)

[Seção 23: E o Business Intelligence? 30](#_Toc78995054)

[Bancos Relacionais x Business Intelligence 30](#_Toc78995055)

[Seção 24: Módulos SQL Server – Elevando o nível 30](#_Toc78995056)

[O client do SQL Server e seus Bancos de Sistema 30](#_Toc78995057)

[A cláusula GO 30](#_Toc78995058)

[Seção 25: Arquitetura do SQL Server 30](#_Toc78995059)

[Arquivos LDF e MDF 30](#_Toc78995060)

[Particionando um banco de dados fisicamente 30](#_Toc78995061)

[Seção 26: Vamos aos Códigos! 30](#_Toc78995062)

[Constraints Nomeadas, Identify, SP\_COLUMNS e SP\_HELP 30](#_Toc78995063)

[A função IFNULL(), Clausula ambígua e a função getdate() 30](#_Toc78995064)

[Trabalhando com Datas 30](#_Toc78995065)

[Conversões de tipos de dados 30](#_Toc78995066)

[A função Charindex 30](#_Toc78995067)

[Utilizando Bulk Insert – Desafio utilizando CHARINDEX 30](#_Toc78995068)

[Correção do desafio – Exibindo o Saldo 30](#_Toc78995069)

[Seção 27: Triggers no SQL Server 30](#_Toc78995070)

[Triggers de DML 30](#_Toc78995071)

[Triggers de DML – Parte 2 30](#_Toc78995072)

[Simplificando! Otimize suas triggers utilizando select 30](#_Toc78995073)

[Triggers de Range – Introdução à Transações 30](#_Toc78995074)

[Seção 28: Programando procedures 30](#_Toc78995075)

[Procedures estáticas, dinâmicas, com parâmetros de entrada e de saída! 31](#_Toc78995076)

[Criando procedures como regra de negócio 31](#_Toc78995077)

[Seção 29: Introdução à linguagem TSQL 31](#_Toc78995078)

[Introdução a TSQL – CAST e CONVERT para padrões da datas 31](#_Toc78995079)

[Atribuindo resultados a variáveis 31](#_Toc78995080)

[Trabalhando com estruturas condicionais – IF ELSE 31](#_Toc78995081)

[Estruturas de loop com While 31](#_Toc78995082)

[Seção 30: Módulo PostgreSQL 31](#_Toc78995083)

[Criando o Primeiro Banco de Dados 31](#_Toc78995084)

[Datas – Trabalhando com Datastyle 31](#_Toc78995085)

[Criando o Banco de Dados do Projeto 31](#_Toc78995086)

[Introdução a Funções de Agregapção 31](#_Toc78995087)

[Média 31](#_Toc78995088)

[Principais Medidas Estatísticas 31](#_Toc78995089)

[Concluindo a Análise Estatística 31](#_Toc78995090)

[Modelagem de Banco de Dados x Modelagem para Data Science 31](#_Toc78995091)

[Importando Arquivos e Verificando a MODA 31](#_Toc78995092)

[Amplitude de um Set de Dados 31](#_Toc78995093)

[Desvio Padrão e Variância 31](#_Toc78995094)

[Mediana 31](#_Toc78995095)

[Coeficiente de Variação 31](#_Toc78995096)

[Moda 31](#_Toc78995097)

[Export Formato Colunar 31](#_Toc78995098)

[Arquitetura do Ambiente 31](#_Toc78995099)

[Entendendo a Estrutura e Organização 31](#_Toc78995100)

[Programando a Sincronização 31](#_Toc78995101)

[Sincronizando Registros Deletados 31](#_Toc78995102)

[Exercício – Salários 31](#_Toc78995103)

[Machine Learning – Criando colunas Dummy 31](#_Toc78995104)

[Introdução aos filtros 31](#_Toc78995105)

[Filtros de contadores 31](#_Toc78995106)

[Formatando Strings 31](#_Toc78995107)

[Seção 31: Módulo Oracle Express 31](#_Toc78995108)

[Instalação e criação de conexões no SQL Developer 32](#_Toc78995109)

[Seção 32: Degustação Módulo Oracle 11g – CONTEÚDO TEMPORÁRIO 32](#_Toc78995110)

[Sobre o Módulo Oracle 32](#_Toc78995111)

[Ambiente Geral Oracle 32](#_Toc78995112)

[Estruturas Lógicas 32](#_Toc78995113)

[Tablespaces 32](#_Toc78995114)

[Estruturas Físicas 32](#_Toc78995115)

[Archive e ASM 32](#_Toc78995116)

[Arquivos de Parâmetros 32](#_Toc78995117)

[Trace e PSWD 32](#_Toc78995118)

[Default Tablespace 32](#_Toc78995119)

[Estruturas de Memória – Parte 01 32](#_Toc78995120)

[Entendendo o Redolog 32](#_Toc78995121)

[Estrutura de PGA 32](#_Toc78995122)

[Configurando Processos 32](#_Toc78995123)

[Analisando Queries no Oracle 32](#_Toc78995124)

[Utilitários Linus e Windows 32](#_Toc78995125)

[Criando o Servidor 32](#_Toc78995126)

[Snapshot e configuração de Boot 32](#_Toc78995127)

[Instalando o Oracle Linux 32](#_Toc78995128)

[Mídias do Oracle Linux 32](#_Toc78995129)

[Conexões de Rede 32](#_Toc78995130)

[Conectando-se ao Linux 32](#_Toc78995131)

[Configurando o Linux 32](#_Toc78995132)

[Configurando o Linux – Parte 2 32](#_Toc78995133)

[Configurações Finais 32](#_Toc78995134)

[Criando o primeiro Snapshot 32](#_Toc78995135)

[Mídias do Oracle 11g 32](#_Toc78995136)

[Xming 32](#_Toc78995137)

[Configurando o Xming 32](#_Toc78995138)

[Instalando o SGBD 32](#_Toc78995139)

[Snapshot de SGBD 32](#_Toc78995140)

[Criando o Banco de Dados 32](#_Toc78995141)

[O Enterprise Manager 32](#_Toc78995142)

[O ambiente windows 33](#_Toc78995143)

[Configurando o servidor Windows 33](#_Toc78995144)

[Instalando o SGBD 33](#_Toc78995145)

[Criando o Banco de Dados 33](#_Toc78995146)

[Criando as variáveis 33](#_Toc78995147)

[Instalando as Ferramentas 33](#_Toc78995148)

[Dicionário de Dados 33](#_Toc78995149)

[Tablespace e Tabelas 33](#_Toc78995150)

[Formatando Colunas 33](#_Toc78995151)

[Configurando o SQL Developer 33](#_Toc78995152)

[Customizando Tablespaces 33](#_Toc78995153)

[Sequences 33](#_Toc78995154)

[Alterando Tablespaces 33](#_Toc78995155)

[Pseudo Colunas 33](#_Toc78995156)

[Triggers – Parte 01 33](#_Toc78995157)

[Triggers – Parte 02 33](#_Toc78995158)

[Triggers – Parte 03 33](#_Toc78995159)

[Triggers – Parte 04 33](#_Toc78995160)

[Operações com Views 33](#_Toc78995161)

[Deferrable Constraints – Parte 01 33](#_Toc78995162)

[Deferable – Parte 02 33](#_Toc78995163)

[Deferable – Parte 03 33](#_Toc78995164)

[Seção 33: Introdução ao BIG DATA 33](#_Toc78995165)

[Introdução – parte 01 33](#_Toc78995166)

[Introdução – parte 02 33](#_Toc78995167)

[Introdução – parte 03 33](#_Toc78995168)

[Introdução – parte 04 33](#_Toc78995169)

[Carreiras de Data Science 33](#_Toc78995170)

[Big Data Real Time e Cloud Computing 33](#_Toc78995171)

[O que é Big Data – Parte 01 33](#_Toc78995172)

[O que é Big Data – Parte 02 33](#_Toc78995173)

[O que é Big Data – Parte 03 33](#_Toc78995174)

[O que é Big Data – Parte 04 33](#_Toc78995175)

[O que é Big Data – Parte 05 33](#_Toc78995176)

[Ciclo de Vida de Business Intelligence 33](#_Toc78995177)

# Seção 1

(Steps de instalação do MySQL e outros programas auxiliares como editores de código (Notepad++ e ...) e softwares de modelagem de bancos de dados (brModelo e StarUML))

# Seção 2: Um pouco da teoria sem esquecer da prática

## Arquitetura de Software

- Padrão MVC (Model, View, Controller): É um padrão-esquema que ilustra como funcionam os componentes de um banco de dados.

View: Aquilo que o usuário do banco de dados vê, ou seja, o que o desenvolvedor front-end se ocupa. As tecnologias tipicamente utilizadas são HTML, CSS e JavaScript

Controller: Parte do processamento das regras de negócio (RN). Onde os dados são processados de acordo com as RN. Tecnologias diversas são utilizadas: Java, Ruby, PHP, C#, Python, etc.

Model: Modelagem do banco de dados, feita sob o formato de várias tabelas (que podem ou não se relacionar, a depender do tipo de banco de dados)

## Modelagem Lógica e Física

- Todo banco de dados surge de um projeto, centralizado em um documento de requisitos (com descrição e detalhamento de quem são os stakeholders, as funcionalidades, integrações c/ outros sistemas, etc.)

Obs: A linguagem SQL é única, mas a sintaxe muda de SGBD (sistema gerenciador de banco de dados) para SGBD. Muda sutilmente, mas muda.

- Processos de Modelagem:

1- Modelagem conceitual: Etapa primária de definição daquilo que o banco de dados deverá armazenar (ex: deve armazenar os dados de identificação do cliente). Pode ser feito como um rascunho em um programa de texto qualquer, a importância dessa etapa será a definição do uso do banco de dados.

2- Modelagem lógica: Etapa de montagem de esquema de banco que contempla as especificidades definidas na etapa de modelagem conceitual, ilustrando os devidos relacionamentos entre entidades e atributos de cada entidade. Requer uso de um software específico como o brModelo, StarUML ou o MySQL Workbench.

3- Modelagem física: Construção dos scripts de banco de dados. É basicamente a etapa em que se coloca em formato de código tudo aquilo que foi definido na etapa conceitual e ilustrado na etapa lógica. Normalmente utiliza-se um editor de texto como Notepad++ ou Sublime text (esse é um pouco mais completo que o notepad++ e indicado para desenvolvimento).

Curiosidade: As etapas 1 (modelagem conceitual) e 2 (modelagem lógica) são realizadas por um Administrador de Dados (AD)

## Modelagem Física

Criando database:

**create** **database** projeto**;**

Usando database

**use** projeto**;**

Criando tabela dentro da database

**create** **table** cliente**(**

Nome varchar**(**30**),**

Sexo char**(**1**),**

Email varchar**(**30**),**

CPF int**(**11**),**

Telefone varchar**(**30**),**

Endereço varchar**(**30**)**

**);**

Verificando/acessando as tabelas do banco de dados

show tables**;**

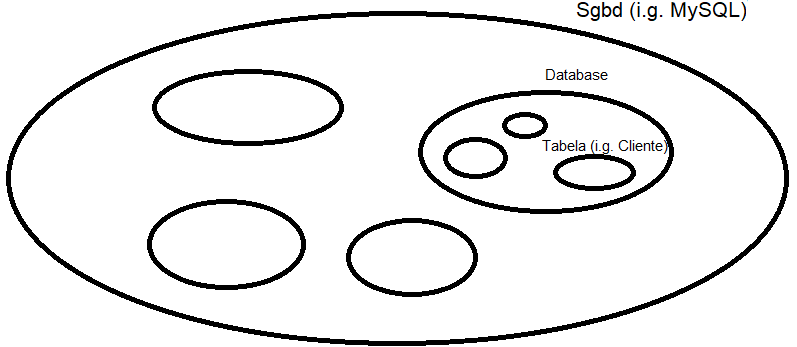
Obs: É possível acessar as tabelas de bancos em que vc não está utilizando, basta especificá-lo no comando

show tables **in** OutraDatabase**;**

Acessando a estrutura de uma tabela

**desc** cliente**;**

Obs:



Um SGBD contém databases (no exemplo acima, criamos a database projeto). Cada database pode conter uma ou mais tabelas (no exemplo, criamos a tabela cliente).

- Observação: O MySQL acompanha o padrão do sistema operacional onde está instalado, ou seja, ele pode ser case sensitive ou não, depende do s.o. onde está instalado (no windows, por exemplo, não há diferenciação entre maiúsculas e minúsculas enquanto no linux há.).

## Comparando CHAR e VARCHAR

- Tipos de dados e correspondentes classificações no SQL

Literais (“strings”) = char, varchar

Números = float, int

Documentos e fotos = blob

Textos extensos = text

- O char é mais performático que o varchar pois é um tipo de dado que não varia (a entrada de dados do usuário deve corresponder exatamente ao número de bytes definido na criação do campo tipado com char). Dessa maneira, sua busca tende a ser mais rápida em campos cujo uso de memória é constante (logo, utiliza-se char para campos de tamanho definido, como sexo, cpf, etc.). Se o campo puder ser preenchido com “tamanhos” de resposta diferentes (como nome, rua, email, etc.), então a regra é usar o varchar na tipagem desses campos.

Obs1:No SQL, é preciso tipar os dados que serão inseridos dentro de tabelas. Eles usualmente são de um dos seguintes tipos: char, varchar, int, float, enum, etc

Obs2: O tipo Enum (que só aceita como entrada um valor de um conjunto definido previamente) só existe no MySQL. Há ainda diferentes tipos de int (big int, small int, tiny int, etc. por isso é necessário consultar a documentação antes de tipar campos com esse tipo de dado).

## Comparando ENUM e Numéricos

- Enum = Tipagem de campo que já possui valores únicos preestabelecidos (como a lista suspensa do excel). Como já foi mencionado, é um tipo que só existe no MySQL, sendo que em outros bancos essa tipagem é definida a partir de constraints.

- Tipos numéricos

Float = Obedece o padrão float(total de dígitos, dígitos após a vírgula). Por exemplo, o float 1505.22 entraria em um campo tipado com, no mínimo, float(6,4). Já o número 2.5 entraria em um campo tipado no mínimo com float(2,1)

Int = É um número inteiro de até 11 dígitos (atentar ao range de tipo de inteiros, como int, bigint, smallint, mediumint, tinyint, etc.). Para mais informações, consultar na documentação do SGBD.

Obs: Em tabelas de SQL, cada coluna é chamada por campo, enquanto cada linha é chamada por registro ou tupla.

# Seção 3: Elevando o nível – Comandos que você não pode deixar de saber!

## Tipos Nulos e Inteiros

Via de regra, só definimos um tipo inteiro quando há intenção de uso para contas matemáticas. Caso contrário, definí-lo como tipo char ou varchar.

Um exemplo bom disso é o CPF e o Telefone, que apesar de serem números, não são utilizados para contas matemáticas, então devem ser tipados como char ou varchar para evitar conflitos desnecessários.

## Conhecendo as Projeções

- Projeção: É o comando select do SQL, que mostra (projeta) algo específico de uma tabela.

Por exemplo, se um gestor quer saber os nomes, sexo e email dos clientes, você precisa mostrar uma projeção que contenha esses campos dos clientes:

**select** nome**,** sexo**,** email **from** cliente**;**

Outro exemplo de projeção utilizando alias

**select** nome **as** cliente**,** sexo**,** email **from** cliente**;**

Obs: Alias é um nome “apelido” que vc dá em uma projeção a um (ou mais) determinado campo. Isso porque muitas vezes o nome que foi dado no campo (na criação da tabela) não é um nome muito descritivo, ou que valem a pena ser reproduzidos para os fins da projeção. Nas queries, é usado por meio do termo as. No primeiro exemplo, a coluna foi tipada com o identificador nome, mas o alias cliente é o que aparece no resultado da query.

Obs2: Evitar utilizar o ‘select \* from [tabela]’ porque desperdiça muito desempenho do banco de dados, já que muitas informações puxadas (o comando puxa literalmente tudo que tem na tabela buscada) não são de fato aproveitadas. Uso para fins acadêmicos, entretanto, é aceitável.

## Seleções

- Seleção é o filtro dos registros através da cláusula where:

**select** nome **from** cliente **where** sexo **=** "m"**;**

O comando acima busca os nomes dos registros cujo sexo foram marcados com m.

- Seleção com comando like:

exemplo:

**select** nome**,** endereço

**from** cliente

**where** endereço **like** 'rj'

No caso acima, o comando like funciona como se fosse um =, mas para strings. O comando irá selecionar nome e endereço dos registros da tabela de cliente, tais que o endereço seja exatamente igual a “rj”

Obs: Caso queira-se selecionar campos que comecem, terminem ou simplesmente contenham uma determinada string, ‘rj’ nesse exemplo, (ou seja, não é exatamente igual ao termo buscado mas que esse termo apareça em algum lugar do registro), utilizamos o caracter coringa % dentro da string.

**...** endereço **like** '%rj' registros que terminam com ‘rj’

**...** endereço **like** '%rj%' registros que contenham ‘rj’

**...** endereço **like** 'rj%' registros que comecem com ‘rj’

Obs2: Seleção aumenta o consumo de desempenho e fluxo de dados dentro de um servidor.

# Seção 4: Exercícios de revisão

Sessão dedicada a um exercício básico de construção de banco de dados com adição de elementos e queries básicas de projeção.

Objetivo: criar tabela simples com informações de livros, p buscas simples + buscas específicas

Etapa 1: Criar e acessar a database livraria

**create** **database** livraria**;**

**use** livraria**;**

Etapa 2: Criar tabela livros:

**create** **table** livro**(**

NomeDoLivro varchar**(**100**),**

NomeDoAutor varchar**(**100**),**

SexoDoAutor char**(**1**),**

NumeroDePaginas int**(**5**),**

NomeDaEditora varchar**(**100**),**

EstadoDaEditora char**(**2**),**

AnoPublicação int**(**4**)**

**);**

Etapa 3: Inserir registros

**insert** **into** livro**(**NomeDoLivro**,** NomeDoAutor**,** SexoDoAutor**,** NumeroDePaginas**,** NomeDaEditora**,** ValorDoLivro**,** EstadoDaEditora**,** AnoPublicacao**)**

**values(**"Cavaleiro Real"**,** "Ana Claudia"**,** "F"**,** 465**,** "Atlas"**,** 49.9**,** "RJ"**,** 2009**);**

Obs: Logicamente esse foi apenas um exemplo de adição de valores dentre os outros que foram adicionadosm, com o objetivo de recordar a sintaxe de adição de valores. Lembrando que não é necessário passar os campos da tabela no comando. A vantagem de colocar os campos junto à tabela, é que o sgbd preenche os campos que não receberam valores com NULL, ao invés de gerar um erro.

Etapa 4: Buscas solicitadas

a) Trazer todos os dados

**select** **\*** **from** livro**;**

**select** NomeDoLivro**,** NomeDoAutor**,** SexoDoAutor**,** NumeroDePaginas**,** NomeDaEditora**,** ValorDoLivro**,** EstadoDaEditora**,** AnoPublicacao

**from** livro**;**

Obs: Primeiro comando traz todo o conteúdo de qualquer tabela (mas só deve ser utilizada para fins acadêmicos)

b) Trazer o nome do livro e o nome da editora

**select** NomeDoLivro**,** NomeDaEditora

**from** livro

c) Trazer o nome do livro e a UF dos livros publicados por autores do sexo feminino

**select** NomeDoLivro**,** EstadoDaEditora

**from** livro

**where** sexo **=** "f"**;**

d) Trazer o nome do livro e o número de páginas dos livros publicados por autores do sexo feminino

**select** NomeDoLivro**,** NumeroDePaginas

**from** livro

**where** sexo **=** "f"**;**

e) Trazer os valores dos livros das editoras de São Paulo

**select** valordolivro **from** livro **where** UF **=** "SP"**;**

Obs: a sintaxe do SQL não é case sensitive, ou seja, tanto faz se trata-se de string, variável, nome de tabela, etc, não faz diferença o uso de caixa alta ou baixa.

f) Trazer todos os dados dos autores do sexo masculino que tiveram livros publicados por São Paulo ou Rio de Janeiro

**select** NomeDoLivro**,** NomeDoAutor**,** SexoDoAutor**,** NumeroDePaginas**,** NomeDaEditora**,** ValorDoLivro**,** EstadoDaEditora**,** AnoPublicacao **from** livro **where** sexo **=** "m" **and** **(**EstadoDaEditora **=** "SP" **or** EstadoDaEditora **=** "RJ"**);**

Obs: Outra maneira de escrever a condição da seleção dessa query seria:

(...) where sexo = “m” and EstadoDaPublicacao in (“SP”, “RJ”));

Essa escrita em tupla é uma outra maneira de escrever consecutivos operadores lógicos do tipo or.

# Seção 5: Lógica de Predicados

## A Tabela Verdade

É a tabela que resolve qualquer operação lógica, em todos os cenários possíveis (considerando que cada proposição pode assumir apenas verdadeiro ou falso). O número de linhas da tabela obedece a fórmula 2^n, sendo n o número de proposições diferentes envolvidas na operação lógica.

Exemplo:

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | A^B | AvB |
| v | v | v | v |
| v | f | f | v |
| f | v | f | v |
| f | f | f | f |

Obs: A^B = A and B

AvB = A or B

Obs2: Uma boa prática de tunning é sempre lembrar de organizar a query de acordo com o menor uso de processamento dos dados. Por exemplo, ao realizar uma query que utiliza o operador OR (e sabendo que basta uma condição ser verdadeira para a expressão ser verdadeira), então convém organizar o primeiro termo como sendo o mais numeroso dentro do bd.

Se, por exemplo, queremos buscar informações dos livros de SP ou do RJ, e sabemos que 70 livros foram publicados em SP e 10 livros publicados no RJ, então a expressão lógica que utiliza menos processamento do banco de dados seria: (...) where EstadoDaEditora = “SP” or EstadoDaEditora = “RJ”, pois basta apenas a primeira verificação (poupando a segunda verificação) para o resultado da expressão ser verdadeiro.

Para expressão lógica AND, a lógica é a mesma, mas no sentido invertido. A condição de menos ocorrência vem em primeiro lugar, pois descarta-se verificações desnecessárias.

## Operadores Lógicos – Prática

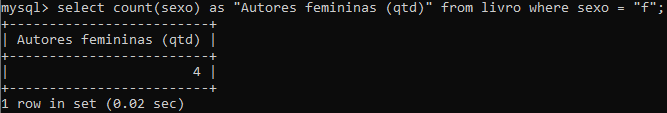
Ou (or) -> Basta uma condição ser verdadeira para a expressão lógica (ou saída) ser verdadeira, caso contrário, é falsa

E (and) -> Todas as condições precisam ser verdadeiras para a expressão lógica ser verdadeira, caso contrário, é falsa

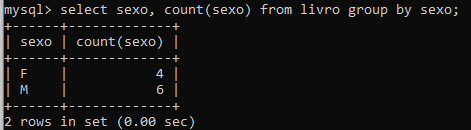
## Performance com Operadores Lógicos

Função de agregação -> count()

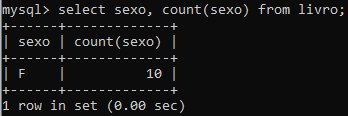
É uma função que traz a quantidade de registros da coluna passada como argumento que se enquadram em uma determinada condição (especificada na query). Exemplo:



Obs: Quase sempre deve-se utilizar um agrupador (group by) ao utilizar funções de agregação. Exemplo:



Obs: Ao não utilizar o comando group by, o banco de dados irá apenas contar os registros que possuem algum valor registrado no campo sexo. Retornará, portanto, 10 (pq são 10 registros nessa tabela, com registros não nulos no campo sexo)



# Seção 6: Exercícios de Fixação

# Seção 7: Mais alguns comandos básicos

## Filtrando Valores Nulos

- Utilizar o ‘is’ para projetar valores nulos (NULL). Exemplo:

**select** nome

**from** funcionarios

**where** email **is** **NULL;**

Obs: Se ao invés de ‘is’ usarmos um =, a query retornará vazia (empty set), pois não é possível igualar NULL a NULL no SQL., por isso é utilizado o operador is

Obs2: Para projetar valores não nulos, basta utilizar a negação (NOT) do operador is. Exemplo:

**select** nome

**from** funcionarios

**where** email **is** **not** **NULL;**

## A cláusula Update

- Serve para atualizar registros da tabela. NUNCA ESQUECER que deve ser usado junto com uma condição de seleção (where) bem definida. Caso contrário, a atualização será replicada em todos os registros da tabela. Exemplo:

**update** funcionarios

**set** email **=** 'jose@unidados.com.br'

**where** nome **=** "Jose"

**and** cpf **=** 8349430912**;**

No exemplo acima, todos os registros da tabela funcionarios com valores de “Jose” no campo nome e valor 8349430912 no campo cpf serão atualizados no campo email para ‘jose@unidados.com.br’

## A cláusula DELETE

- Assim como a cláusula update, a cláusula delete deve ser utilizada junto a uma condição bem definida de seleção (where). Caso contrário, deletará todos os registros da tabela. Exemplo:

**delete** funcionarios

**where** nome **=** "Carlos" **and** email **=** "carlos@unidados.com.br"**;**

No exemplo acima, o comando deletará da tabela funcionarios todos os registros que na coluna nome recebem o valor ‘Carlos’ e na coluna email recebem o valor ‘carlos@unidados.com.br’.

Dica importante: Antes de utilizar qualquer comando de DTL (como delete, update), convém fazer uma projeção da query para conferir se os valores projetados batem com aqueles valores que se deseja atualizar ou deletar da tabela.

# Seção 8: Modelando Bancos de Dados para Sistemas

## Começando a Modelar

- Campos vetorizados são aqueles que comportam mais de um dado. Por exemplo, endereço é um campo vetorizado, pois há mais de um tipo de dado incluso, como rua, , número, bairro, cidade, estado, complemento, etc. É campo vetorizados porque todos eles pertencentes a uma mesma categoria.

Dessa maneira, dentro de bancos relacionais, todo e qualquer campo multivalorado ou vetorizado se tornará obrigatoriamente uma nova tabela.

Obs: Há uma pequena distinção semântica entre campos vetorizados e campos multivalorados. Campos vetorizados são aqueles que comportam múltiplos valores, mas todos eles pertencentes a uma mesma categoria (exemplo do endereço acima), já campos vetorizados comportam múltiplos dados mas sem relação entre si.

## A história da Modelagem

- Não modelar bancos de dados orientados a processos e regras gerais. A orientação de qualquer modelagem de bancos de dados deve ser sempre a regras de negócio.

## Primeira Forma Normal

1ª Regra: Todo campo vetorizado se tornará outra tabela

2ª Regra: Todo campo multivalorado se tornará outra tabela

3º Regra: Toda tabela necessita de pelo menos um campo que identifique o registro como único (chave primária)

Obs: Não há consenso sobre como é chamado o campo que contém a primary key da tabela, mas usa-se com frequência a terminologia “id” + “nome da tabela”. Por exemplo, a pk de uma tabela cliente seria idcliente.

## Cardinalidade e Obrigatoriedade

- Quem define a cardinalidade e obrigatoriedade não é quem cria e modela o banco de dados, e sim **as regras de negócio.**

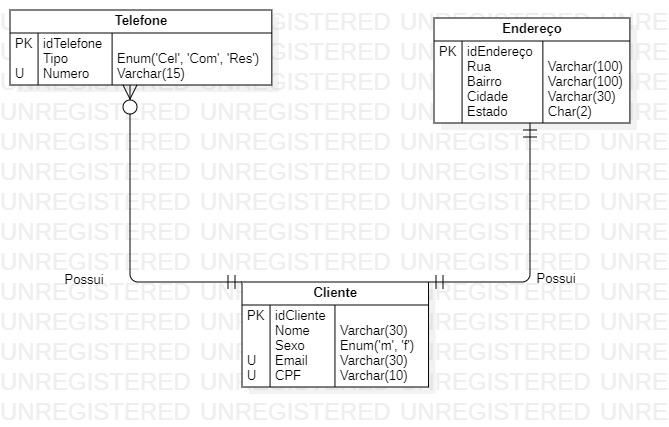
- Obrigatoriedade (0 ou 1) diz respeito se a tabela é obrigatoriamente preenchida ou não, enquanto a cardinalidade diz respeito ao número de registros a serem feitos por entrada no sistema (1 ou n).

- A sintaxe é feita no seguinte modo: (obrigatoriedade, cardinalidade) e cada tabela pode assumir uma configuração diferente, a depender do relacionamento que se pretende com outras tabelas (normalmente se mantém, mas é possível que em casos específicos isso se altere).



A parte esquerda corresponde à obrigatoriedade, enquanto a parte direita corresponde à cardinalidade.

Obs: Essa notação descreve apenas para descrever individualmente cada tabela. Para tratar dos relacionamentos entre tabelas, olha-se apenas para a cardinalidade das duas entidades.



No exemplo acima, a notação de relacionamento entre entidades está feita no formato de pé de galinha (será visto mais pra frente), mas a ideia é a mesma da que foi passada acima. Cliente possui um relacionamento de 1 pra 1 com Endereço (cada entrada de cliente contém uma entrada de cliente), enquanto que o relacionamento de Cliente com Telefone possui relacionamento de 1 pra n (cada entrada de cliente possui n telefones)

Notações de pé de galinha (Vide seção 12):



## Modelo Lógico para Físico – Parte 01

A modelagem física da tabela Cliente (exemplo acima) é a seguinte:

**create** **table** cliente**(**

idCliente int **primary** **key** auto\_increment**,**

Nome varchar**(**30**),**

Sexo Enum**(**'m'**,** 'f'**)**

Email varchar**(**30**)** **unique,**

Cpf varchar**(**10**)** **unique**

**);**

Lembrando que tanto a constraint auto\_increment da primary key quanto o data type enum() são exclusivos do MySQL

- Constraints:

a) Primary Key: reserva o campo como sendo identificador de registros daquela tabela

b) auto\_increment: Faz com que o SGBD se ocupe da numeração incremental do campo (não esquecer de declarar o tipo do campo como inteiro).

c) Unique: garante que entradas naquele campo serão sempre únicas (não haverá repetição de registros)

d) Not null: impede que o SGBD aceite entradas nulas para aquele campo.

## Modelo Físico – Parte 02

Modelagem física das outras duas tabelas:

**create** **table** endereço**(**

idEndereço int **primary** **key** auto\_increment**,**

rua varchar**(**100**),**

Bairro varchar**(**100**),**

Cidade varchar**(**30**),**

Estado char**(**2**)**

**);**

**create** **table** Telefone**(**

idTelefone int **primary** **key** auto\_increment**,**

Tipo enum**(**'com'**,** 'cel'**,** 'res'**)** **not** **null,**

Numero varchar**(**15**)**

**);**

## Entendendo a Foreign Key – Parte 01

- Chave estrangeira (ou FK, de Foreign Key) é a chave primária de uma tabela que vai até outra tabela (e passa a receber o nome de chave estrangeira) para fazer referência entre registros.

- A tabela que conterá a chave estrangeira varia de acordo com os relacionamentos entre tabelas (relacionamento de cardinalidades):

1 x 1 = FK fica na tabela mais fraca (de menor valor estratégico para o negócio)

1 x n = FK fica sempre na tabela de cardinalidade N

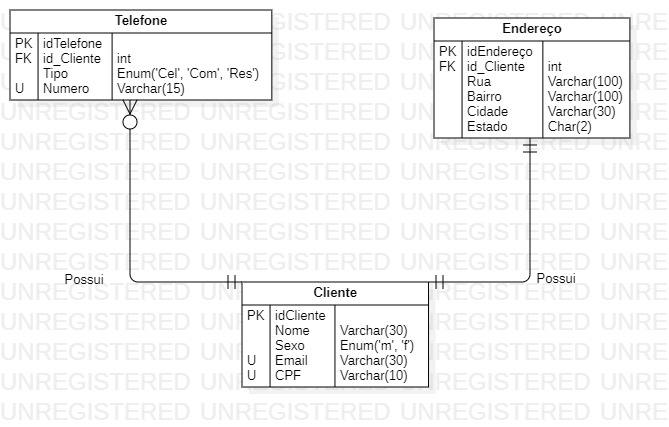
N x n = A FK não fica em nenhuma das tabelas. Cria-se uma outra tabela, chamada de tabela associativa, cujo objetivo é exclusivamente relacionar as PKs das duas tabelas.

-Obs: Chama-se banco de dados relacionais pois as chaves (primárias e estrangeiras) garantem a integridade referencial, ou seja, que registros guardem as referências entre si, mesmo estando armazenados em entidades (tabelas) separadas.

## Foreign Key – Parte 02

- Nomenclatura de chave estrangeira: “id” + “\_” + “tabelaorigem”. Por exemplo, se uma chave estrangeira da tabela telefone faz referência à chave primária da tabela cliente, então seu nome seria id\_cliente (e ficaria dentro da tabela telefone).

- Implementando as chaves estrangeiras no esquema:



- Na modelagem física, a definição das foreign keys é feitas da seguinte forma:

**create** **table** cliente**(**

idCliente int **primary** **key** auto\_increment**,**

Nome varchar**(**30**),**

Sexo Enum**(**'m'**,** 'f'**),**

Email varchar**(**30**)** **unique,**

Cpf varchar**(**10**)** **unique**

**);**

**create** **table** endereço**(**

idEndereço int **primary** **key** auto\_increment**,**

rua varchar**(**100**),**

Bairro varchar**(**100**),**

Cidade varchar**(**30**),**

Estado char**(**2**),**

id\_Cliente int**,**

**foreign** **key(**id\_Cliente**)**

**references** Cliente**(**idCliente**)**

**);**

**create** **table** Telefone**(**

idTelefone int **primary** **key** auto\_increment**,**

Tipo enum**(**'com'**,** 'cel'**,** 'res'**)** **not** **null,**

Numero varchar**(**15**),**

id\_Cliente int**,**

**foreign** **key(**id\_Cliente**)**

**references** Cliente**(**idCliente**)**

**);**

## Serviços do Banco, Password e Sublime Text

(Apresentação básica do Sublime, que é um editor de texto mais apropriado para desenvolvimento de banco de dados)

## Inserindo Dados

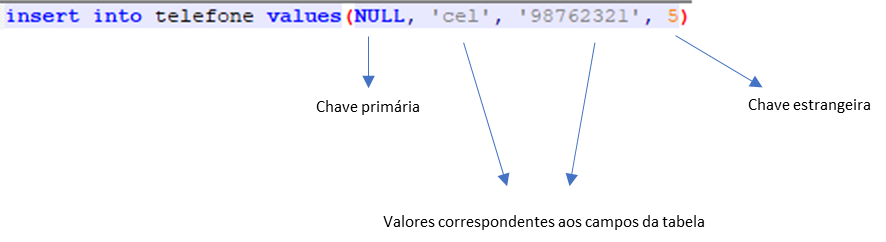
- Em tabelas que possuem chaves primárias (que, no MySQL, foram definidas com a constraint auto\_increment) basta passar o valor NULL dentro da sintaxe de insert into ... values()

## Inserções em Relacionamentos 1 x 1

- Lembrar de colocar a constraint Unique na definição da chave estrangeira, em relacionamentos 1 x 1.

Obs: A FK fica na tabela mais fraca, em relacionamentos 1 x 1.

## Inserções 1 x N



Obs: Passa-se NULL como valor correspondente à chave primária pois a constraint auto\_increment cuida da numeração não repetida. Já a chave estrangeira precisa conter o valor do registro da chave primária a que se deseja referenciar o registro.

# Seção 9: Juntando as peças

## Seleção e Projeção

- Seleção é o processo de filtragem de uma (ou mais) tabela. Formalmente, corresponde a um subconjunto do conjunto total de registros da tabela. Por exemplo, no seguinte código:

**select** nome **from** cliente **where** sexo **=** 'm'**;**

A cláusula where corresponde à seleção, porque filtra a busca apenas dos registros que são do sexo masculino.

- Projeção é o nome que se dá a uma projeção específica de uma (ou mais) tabela. É possível projetar dados extraídos de tabelas ou outras informações externas como data (select now()) ou contas matemáticas (select 1 + 1 as soma;)

Corresponde à cláusula select dos comandos SQL.

- Obs: “Origem” (não é um nome consensual, mas tratemos por origem aqui) não é obrigatório no MySQL (como em select now();), mas em outros SGBD, como Oracle, é.

Exemplo:

**select** nome /\* cláusula de projeção\*/

**from** cliente /\* cláusula de ‘origem’ \*/

**where** sexo **=** 'm'**;** /\*clásula de seleção \*/

## Junção

- Refere-se à cláusula SQL de inner join

- Serve para integrar tabelas (através da combinação das chaves primárias e estrangeiras) na hora de realizar buscas que exijam dados distribuídos em tabelas diferentes. Ex:

**select** nome**,** sexo**,** email**,** tipo**,** numero

**from** cliente

**inner** **join** telefone

**on** idcliente **=** id\_cliente /\* Iguala-se a chave primária com a chave

estrangeira, na ordem de aparição das

tabelas na sintaxe \*/

Obs: Lembrar que em casos de relacionamento 1xn, é esperado que hajam registros “repetidos”, afinal de contas há apenas um tipo de registro em uma das tabelas (de cardinalidade 1) para n registros na outra tabela (de cardinalidade n)

## Inner Join

- Ponteiramento de endereço de memória: É o recurso utilizado nas sintaxes de inner join pois aponta qual campo pertence a qual das tabelas, assim evitando “confusões” do sgbd na hora de retornar as buscas desejadas. Para ponteirar, basta dar um espaço após a nomeação das tabelas com o caracter (ou sequência de caracteres) que se deseja configurar como ponteiros. Esse caracter deve aparecer antes dos campos mencionados ao longo de toda a sintaxe. Ex:

**select** c**.**nome**,** c**.**sexo**,**

e**.**bairro**,** e**.**cidade**,**

t**.**tipo**,** t**.**numero

**from** cliente c

**inner** **join** endereço e

**on** c**.**idcliente **=** e**.**id\_cliente

**inner** **join** telefone t

**on** c**.**id\_cliente **=** t**.**id\_cliente**;**

No comando acima, primeiro definimos os ponteiros das tabelas (a tabela cliente foi ponteirada com c, a tabela endereço foi ponteirada com e, a tabela telefone foi ponteirada com t).

Após o ponteiramento das tabelas, relaciona-se os campos procurados com esses ponteiros.

Dessa maneira, essa sintaxe fará o sgbd retornar nome, sexo, bairro, cidade, tipo e número cruzando dados das tabelas cliente, endereço e telefone.

- Obs: O recurso de ponteiramento também aumenta a performance do banco de dados

## Comandos de DML

- O SQL é uma linguagem de 4ª geração (alto nível) e possui 4 categorias diferentes:

1- DML (Data Manipulation Language):

Ex: insert, select, update, delete, etc.

2- DDL (Data Definition Language)

3- DCL (Data Control Language)

4- TCL (Transaction Control Language)

## DDL – Modificando Tabelas

- Corresponde à etapa de criação das tabelas, onde há a tipagem dos campos que receberão entradas correspondentes e outros comandos de alteração de tipagem (ex: create table, alter table)

Exemplos:

1- Alterando o nome de uma coluna

**alter** **table** produto

change preço valor\_unitario int **not** **null;**

/\* Trocando a coluna preço pelo nome de valor\_unitario, com entradas inteiras e não nulas \*/

Obs: Caso só queira alterar o tipo de dado que uma coluna já criada admite, deve-se redefinir, ainda assim, a coluna inteira (no caso, redefine-se o nome da coluna para ele mesmo seguido da alteração do tipo de dado recebido)

Obs2: Uma maneira mais fácil de realizar a mesma alteração (apenas do tipo de dado recebido) seria com a cláusula modify. Ex:

**alter** **table** produto

**modify** preço float**(**5**,**2**);**

Obviamente o dado que se pretenda mudar deve ser compatível com o dado definido até então. Não é possível transformar uma coluna inicialmente tipada com varchar para int (exceto se só tiverem sido registrados números nessa coluna).

2- Adicionando coluna

**alter** **table** produto

**add** peso float**(**10**,**2**)** **not** **null;**

Por padrão, a coluna adicionada será colocada em último lugar na tabela. Mas é possível escolher o local onde será adicionada a coluna dentro da tabela. Basta especificar no comando adicionando “first” (caso queira colocar como primeira coluna). Ex:

**alter** **table** produto

**add** peso float**(**10**,**2**)** **not** **null** **first;**

Ou então, caso queira adicionar em um lugar específico, utilizar “after” + coluna. Ex:

**alter** **table** produto

**add** peso float**(**10**,**2**)** **not** **null** **after** valor\_unitario**;**

3- Removendo coluna

**alter** **table** produto

**drop** **column** peso**;**

# Seção 10: Exercícios de Fixação

Exercícios de DML

# Seção 11: Aprofundando

## Funções – IFNULL

- A utilidade dessa função é trocar valores nulos que apareçam em uma query por valores específicos (passados como argumento, na chamada da função). Sua chamada ocorre no comando de projeção (após o select). Ex:

**select** c**.**nome**,**

**ifnull(**c**.**email**,** "não registrado"**),**

e**.**estado**,**

t**.**numero

**from** cliente c

**inner** **join** endereço e

**on** c**.**idcliente **=** e**.**id\_cliente

**inner** **join** telefone t

**on** c**.**idcliente **=** t**.**id\_cliente**;**

No comando acima, a função ifnull irá substituir valores nulos registrados na coluna email pela string “não registrado”. Este valor só será retornado na query, não sendo registrado substituindo os valores nulos na tabela.

- A função ifnull existe nos outros bancos de dados (Oracle, SQL Server, Postgree, etc) mas com outras sintaxes

## Views

- São ponteiros de memória para querys. Com a construção de uma view, você não precisa reescrever sempre a mesma sintaxe de uma query para conseguir acessá-la. É útil especialmente em querys complexas, pois economiza tempo gasto. Exemplo de sintaxe:

/\*1- A Query: \*/

**select** c**.**nome**,** c**.**sexo**,** c**.**email**,**

t**.**tipo**,** t**.**numero**,** t**.**bairro**,**

e**.**bairro**,** e**.**cidade**,** e**.**estado

**from** cliente c

**inner** **join** telefone t

**on** c**.**idcliente **=** t**.**id\_cliente

**inner** **join** endereço e

**on** c**.**idcliente **=** e**.**id\_cliente**;**

/\* 2- A View (reproduzindo a query acima) \*/

**create** **view** v\_Relatório **as**

**select** c**.**nome**,** c**.**sexo**,** c**.**email**,**

t**.**tipo**,** t**.**numero**,** t**.**bairro**,**

e**.**bairro**,** e**.**cidade**,** e**.**estado

**from** cliente c

**inner** **join** telefone t

**on** c**.**idcliente **=** t**.**id\_cliente

**inner** **join** endereço e

**on** c**.**idcliente **=** e**.**id\_cliente**;**

/\* 3- Acessando a view \*/

**select** **\*** **from** v\_Relatório**;**

Obs: A View fica armazenada dentro da database como uma nova tabela (é possível verificá-la com o show tables)

Obs2: É boa prática adicionar algum identificador às views. Nesse exemplo (e é um identificador possível), utilizei o v\_ antes do nome Relatório. Assim, em um comando do tipo show tables é possível identificar, sem checagens ulteriores, o que é tabela e o que é view.

Obs3: É possível fazer outras queries ainda mais específicas dentro de uma view (são chamadas de subqueries)

- Apagando uma view:

**drop** **view** v\_Relatório**;**

## Operações de DML em Views

- Insert e delete: Não é possível adicionar ou remover registros de views.

- Update: É possível atualizar registros, mas lembrar que ele será feito na tabela original. Isso acontece pois a view nada mais é que um ponteiro de memória, ou seja, remete necessariamente à tabela original.

## Ordenando Dados

- É feito através do comando **order** **by** e ordena os resultados de uma query de acordo com o nome da coluna (ou índice correspondente). É possível ordenar mais de um campo em uma query.

Exemplo: Suponhamos uma tabela Jogadores com os campos nome e número.

**select** nome

**from** jogadores **order** **by** nome**;**

/\* Query retornará nomes ordenados por ordem alfabética

(padrão do comando order by) \*/

**select** nome

**from** jogadores **by** número

/\* query retornará nomes dos jogadores ordenados pelo número

(ordem crescente é o padrão para campos numéricos) \*/

Obs - Ao invés de dar os nomes das colunas, é possível utilizar índice das colunas (começa em 1). Entretanto, os índices só podem se referir às colunas mencionadas na projeção da query (ou seja, após o comando select). Caso queira-se ordenar a query por uma coluna que não está enunciada na query, então deve-se utilizar o próprio nome da coluna, e não índice.

Obs2 – É possível realizar a ordenação da query de acordo com mais de uma coluna, bastando separá-las por vírgulas

**select** nome

**from** jogadores

**order** **by** nome**,** número

/\* Ordenação primeiro em ordem alfabética (campo nome)

e depois ordenação em ordem crescente (campo número) \*/

Obs3 - Para números, o padrão de ordenação no SQL é ordem crescente e para alfabéticos, o padrão é ordem alfabética. Caso queira-se especificar a ordem inversa, basta utilizar **asc** (crescente) ou **desc** (decrescente) após cada coluna. Ex:

**select** nome

**from** jogadores

**order** **by** nome **asc,** número **desc**

/\* Ordenará o resultado da busca primeiro pelo campo

nome, em ordem crescente, e depois pelo campo número,

em ordem decrescente.

Obs: O primeiro asc é desnecessário, pois já é o padrão,

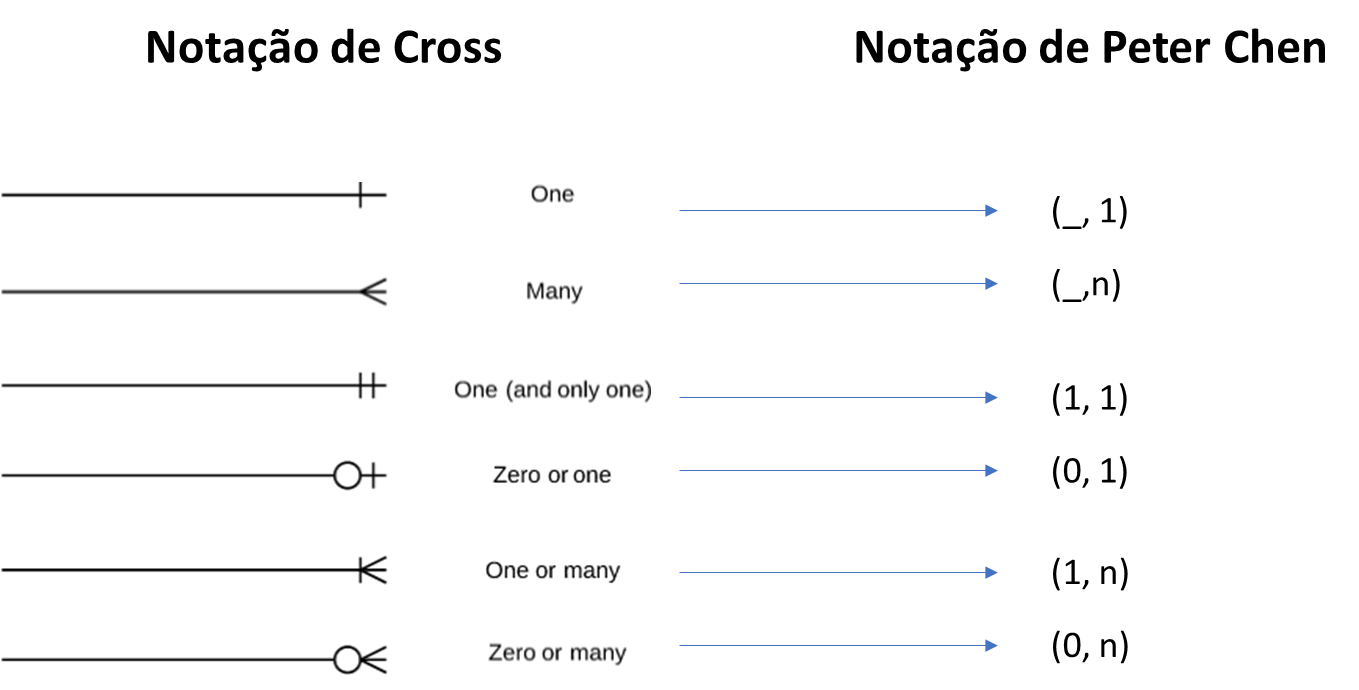
tanto para números quanto para alfabéticos\*/

# Seção 12: Outras Notações

## Notação de Cross e o StarUML

- Notação de Cross é uma maneira de representar os relacionamentos entre entidades dentro da modelagem lógica do banco de dados. A maneira vista até o momento foi a notação de Peter Chen, utilizando pares para cada tabela no formato (obrigatoriedade, cardinalidade).

A notação de Cross representa a mesma coisa, mas de maneira mais gráfica. É também chamada de notação pé de galinha



- Vantagens/Desvantagens: A notação de Peter Chen é mais utilizada dentro do contexto acadêmico enquanto a notação de Cross é mais utilizada no mercado, por arquitetos de dados devido à estética mais limpa e menos poluída, o que facilita no entendimento de projetos mais complexos de modelagem de bancos de dados.

# Seção 13: Programando em MySQL

## Delimiter

- Delimitador é o caracter que separa os blocos de comandos. No SQL, por padrão esse caracter é definido como o ; no fim do comando.

É possível alterar o caracter delimitador através do comando delimiter. Ex:

delimiter $

O delimitador, que por padrão é ; passa a ser o caracter $.

Obs: É possível verificar qual o delimitador configurado (bem como outras informações através do comando status



## Iniciando com Programação em Bancos de Dados

- Procedures são blocos de programação dentro do banco de dados. Na prática, eles funcionam como se fossem funções, criadas pelo usuário do banco de dados, que operam sobre o banco de dados.

- Para criar uma Procedure é imprescindível alterar o caracter delimitador, para que o bloco de programação não seja lido e executado logo na primeira linha do bloco programado.

Ex:

1- Criando a procedure

delimiter $ /\*Altero o delimitador para iniciar a programação da procedure \*/

**create** **procedure** nome\_empresa**()**

**begin**

**select** 'Universidade dos Dados' **as** empresa**;**

**end**

$

delimiter **;** /\*Altero novamente o delimitador para o padrão \*/

2- Chamando a procedure

**call** nome\_empresa**();** /\*Bloco contido na procedure será executado\*/

- As procedures podem conter parâmetros

Exemplo de procedure com parâmetro:

1- Criando a procedure

delimiter $

**create** **procedure** conta**(**número1 int**,** número2 int**)**

**begin**

**select** número1 **+** número2 **as** soma**;**

**end**

$

delimiter**;**

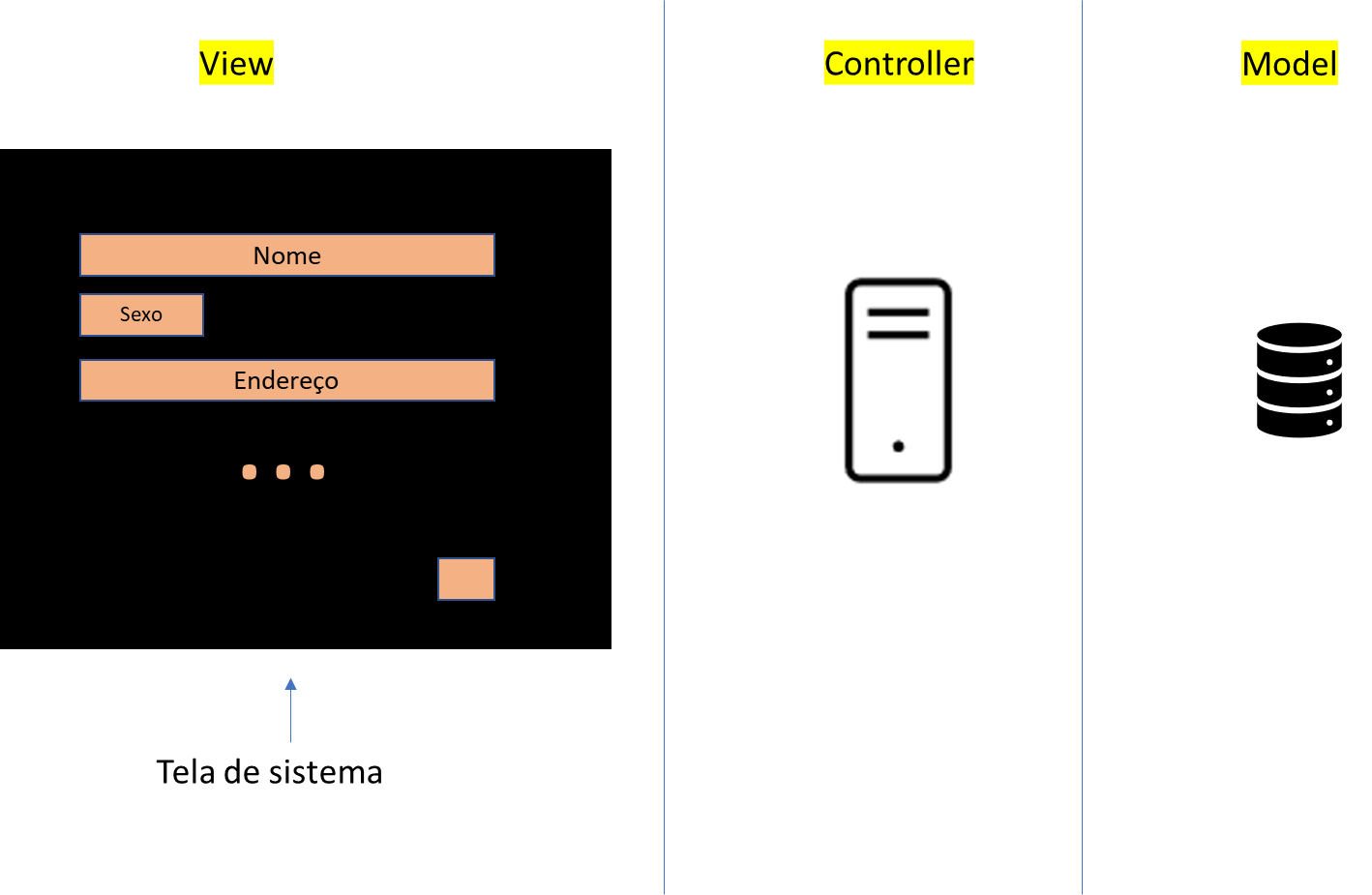
2- Chamando a procedure

**call** conta**(**10**,** 20**);**

## Procedures no Mundo Real

- Normalmente em empresas, a boa prática da gestão de banco de dados sugere que haja um servidor de controle e processamento dos dados inseridos através da tela do sistema (view do padrão MVC), antes de serem diretamente inputados no servidor de banco de dados. Existe a possibilidade de deixar na mão das procedures essa etapa de controle e processamento de dados, mas ao fazer isso, as regras de negócio ficam no servidor de banco de dados, o que dificulta a migração entre bancos de dados (pois cada um tem a sua sintaxe própria, ainda que o SQL seja a linguagem)

- Esquema de banco de dados (Padrão MVC)



1- View: Contém as linguagens de frontend (html, css, javascript)

2- Controller: Servidor dedicado, com processamento de dados de acordo com as regras de negócio (c#, php, java, ruby, javascript, etc)

3- Model: Banco de dados, que armazenará os dados processados. (SQL)

Obs: As procedures podem substituir as linguagens de controle e processamento, sendo posicionadas dentro do próprio servidor de banco de dados. Há vantagens e desvantagens nisso, principalmente relacionadas ao uso de processamento do servidor.

Obs2: O exemplo acima é apenas um exemplo de organização, o servidor de controle pode ser o mesmo de modelagem, e vice versa

## Procedures com Query – Parâmetros

- Criando database de cursos

**create** **database** projeto**;**

**use** projeto**;**

**create** **table** cursos**(**

idcurso int **primary** **key** auto\_increment**,**

nome varchar**(**30**)** **not** **null,**

horas int**(**3**)** **not** **null,**

valor float**(**10**,**2**)** **not** **null**

**);**

- Criando procedure de dml (insert)

delimiter $

**create** **procedure** cad\_curso**(**p\_idcurso int**,** p\_nome varchar**(**30**),** p\_horas int**(**3**),** p\_valor float**(**10**,**2**))**

**begin**

**insert** **into** cursos**(**idcurso**,** nome**,** horas**,** valor**)** **values(NULL,** p\_nome**,** p\_horas**,** p\_valor**);**

**end**

$

delimiter **;**

- Chamando a procedure

**call** cad\_curso**(**'BI SQL Server'**,** 35**,** 2000.00**);**

**call** cad\_curso**(**'Power BI'**,** 20**,** 1000.00**);**

**call** cad\_curso**(**'Tableau'**,** 30**,** 1200.00**);**

Obs: É boa prática adicionar o prefixo “p\_” antes dos parâmetros, na definição da procedure. Evita que o sgbd dê erros por não saber se a referência é aos parâmetros passados ou aos campos da tabela.

- Questão desafio: Faça uma procedure que busque o curso pelo nome e traga todas as informações dele

delimiter $

**create** **procedure** procura**(**p\_nome varchar**(**30**))**

**begin**

**select** idcurso**,** nome**,** horas**,** valor **from** cursos **where** nome **=** p\_nome**;**

**end**

$

delimiter **;**

# Seção 14: Agregue, some, me dê a média e tudo mais – Funções de Agregação

## Group by, count, max, min, avg e funções do MySQL

Suponhamos a seguinte tabela:

**create** **table** Vendedores**(**

idvendedor int **primary** **key** auto\_increment**,**

nome varchar**(**30**),**

sexo char**(**1**),**

janeiro float**(**10**,**2**),**

fevereiro float**(**10**,** 2**),**

março float**(**10**,** 2**)**

**);**

Funções úteis:

1- Max(): Retorna o valor máximo de uma coluna

**select** **max(**janeiro**)** **as** max\_jan **from** vendedores**;**

2- Min(): Retorna o valor mínimo de uma coluna

**select** **min(**fevereiro**)** **as** min\_fev **from** vendedores**;**

3- Avg(): Retorna o valor médio de uma coluna

**select** **avg(**marco**)** **as** avg\_mar **from** vendedores**;**

4- ... (Existem outras funções semelhantes, como o std, etc)

## 

## Tudo certo, mas agora eu quero a soma – Utilizando o SUM!

- Função sum() recebe uma coluna como argumento e retorna a soma de todos os valores

**select** sexo**,** **sum(**marco**)** **as** total\_marco **from** vendedores **group** **by** sexo**;**

- sum() x count(): A função sum() retorna a soma dos valores da coluna, enquanto a função count() conta quantos valores não nulos existem.

# Seção 15: Subconjunto do Conjunto

## Utilizando Subqueries

## Somei as colunas. Mas e as linhas? Operações aritméticas.

# Seção 16: Organizando a casa – O dicionário de dados!

## Verificando e alterando a estrutura de uma tabela, seus objetos e charsets

## Organizando as chaves e introdução às bases de dicionário

## Constraints Nomeadas x Dicionário de Dados

## Aprofundando com constraints – Querys de Dicionários

# Seção 17: Vamos Revisar?

## Revisão ilustrada sobre Foreign Keys

# Seção 18: Aprendeu? Saiu o sistema do Seu José

# Seção 19: Automatizando as coisas – Triggers

## Introdução às fantásticas Triggers

- Trigger é um gatilho programado. Uma ação programada para ser realizada antes (ou depois) de outra ação.

- Sintaxe:

**create** **trigger** nome

**after/before** **insert/delete/update** **on** tabela

**after** **each** **row**

**begin**

bloco de comandos **SQL**

**end**

- Função da trigger: Realiza uma ação (especificada no bloco de comandos SQL) dado um determinado evento a ser especificado na sintaxe de criação da trigger.

Obs: Lembrar de alterar o delimitador antes de definir a procedure (exatamente como na definição da procedure).

Obs2: Triggers são muito utilizadas por DBAs para monitorar e aumentar o controle sobre os bancos de dados.

## Agora é pra valer! Trigger na prática

- Criando uma trigger de backup de tabela:

1- Suponhamos a tabela original usuario(idusuario, nome, login, senha) e a tabela de backup de usuario, bkp\_usuario(idbkpusuario, idusuario, nome, login)

2- Criamos a trigger de backup da tabela usuario

delimiter $

**create** **trigger** bkp\_user

**after** **delete** **on** usuario

**begin**

**insert** **into** bkp\_usuario **values(null,** **old.**idusuario**,** **old.**nome**,** **old.**login**);**

**end**

$

delimiter **;**

- Old / New: São palavras reservadas do SQL para se referirem às triggers. Funciona segundo a lógica temporal, ou seja, um valor old é aquele que foi removido (delete) ou substituído (update) enquanto um valor new é um valor que foi adicionado (insert) ou substituto (update). Prestar atenção em qual categoria (old ou new) descreve realmente o valor que vc deseja.

## Triggers para bancos de backups

- Comunicação entre bancos de dados para backup: Um jeito de alternar databases (em triggers, procedures, etc) é usar a notação database.table

Dessa forma, o SGBD não perde a referência das tabelas que ficam em outras databases, quando referenciadas no meio de um comando.

Isso é importante pois normalmente manipula-se databases de cópia (backup), onde são salvas as ações realizadas nas tabelas da database original.

- Vamos tomar como exemplo a criação de triggers (e manipulação de diferentes bancos de dados) em um exemplo de tabela produto.

1- Cria-se as tabelas

**CREATE** **DATABASE** LOJA**;**

**USE** LOJA**;**

**CREATE** **TABLE** PRODUTO**(**

IDPRODUTO INT **PRIMARY** **KEY** AUTO\_INCREMENT**,**

NOME VARCHAR**(**30**),**

VALOR FLOAT**(**10**,**2**)**

**);**

**CREATE** **DATABASE** **BACKUP;**

**USE** **BACKUP;**

**CREATE** **TABLE** BKP\_PRODUTO**(**

IDBKP INT **PRIMARY** **KEY** AUTO\_INCREMENT**,**

IDPRODUTO INT**,**

NOME VARCHAR**(**30**),**

VALOR FLOAT**(**10**,**2**)**

**);**

**USE** LOJA**;**

Acima temos a database loja e a database backup. A database loja possui a tabela produto, e a database backup tem a tabela bkp\_produto. Ou seja, uma trigger de backup de DMLs na tabela original envolveria, obrigatoriamente, mudança de databases. Fazemos ela da seguinte maneira:

DELIMITER $

**CREATE** **TRIGGER** BACKUP\_PRODUT

**BEFORE** **INSERT** **ON** PRODUTO

**FOR** **EACH** **ROW**

**BEGIN**

**INSERT** **INTO** **BACKUP.**BKP\_PRODUTO **VALUES(NULL,NEW.**IDPRODUTO**,**

**NEW.**NOME**,NEW.**VALOR**);**

**END**

$

DELIMITER **;**

Na trigger definida acima, qualquer valor que for adicionado na tabela Produto (loja.produto) será antes adicionada à tabela bkp\_produto (backup.bkp\_produto)

- Observação importante: Dentro desse tipo de estratégia (chamada de backup lógico), é possível inserir várias triggers para alimentar a mesma tabela da database de backup, no entanto, elas precisam “cobrir” (ou especificar) momentos diferentes de uma mesma operação de DML. Ou seja, não dá pra fazer duas triggers diferentes cuidarem do **AFTER** **INSERT** **ON** PRODUTO

Exemplo de outra trigger, que irá fazer o backup após deletes na tabela produto:

DELIMITER $

**CREATE** **TRIGGER** BACKUP\_PRODUTO\_DEL

**BEFORE** **DELETE** **ON** PRODUTO

**FOR** **EACH** **ROW**

**BEGIN**

**INSERT** **INTO** **BACKUP.**BKP\_PRODUTO **VALUES(NULL,OLD.**IDPRODUTO**,**

**OLD.**NOME**,OLD.**VALOR**);**

**END**

$

DELIMITER **;**

Ps: A trigger backup\_produt foi criada com o nome errado, então vamos deletar com o comando drop para recriá-la com o nome correto (com a correção lógica do after ao invés do before, pois o before faz com que o idproduto não seja registrado corretamente na tabela de backup, já que ele será registrado antes da chave primária assumir o valor)

**DROP** **TRIGGER** BACKUP\_PRODUT**;**

DELIMITER $

**CREATE** **TRIGGER** BACKUP\_PRODUTO

**AFTER** **INSERT** **ON** PRODUTO

**FOR** **EACH** **ROW**

**BEGIN**

**INSERT** **INTO** **BACKUP.**BKP\_PRODUTO **VALUES(NULL,NEW.**IDPRODUTO**,**

**NEW.**NOME**,NEW.**VALOR**);**

**END**

$

DELIMITER **;**

## After, before, insert, delete ou update? Os eventos de uma trigger

- Uma maneira inteligente e prática de entender qual evento cada linha da tabela de backup se refere é adicionar uma coluna dedicada ao tipo, tanto na criação da tabela de backup quanto na própria criação da trigger. Assim, na trigger de backup de delete, adiciona-se o valor ‘d’, enquanto na trigger de backup de insert adiciona-se ‘i’ e na trigger de backup de update adiciona-se ‘u’.

Exemplo (trigger de backup de delete):

DELIMITER $

**CREATE** **TRIGGER** BACKUP\_PRODUTO\_DEL

**BEFORE** **DELETE** **ON** PRODUTO

**FOR** **EACH** **ROW**

**BEGIN**

**INSERT** **INTO** **BACKUP.**BKP\_PRODUTO **VALUES(NULL,OLD.**IDPRODUTO**,**

**OLD.**NOME**,OLD.**VALOR**,**'D'**);**

**END**

$

DELIMITER **;**

## Quem mexeu no meu dado? Auditando uma tabela com trigger

Exemplo de auditoria completa (com informações de quem alterou e quando alterou a tabela):

1- Criação da tabela de backup, com os devidos campos de identificação de autoria/data e informações

**CREATE** **DATABASE** **BACKUP;**

**USE** **BACKUP;**

**CREATE** **TABLE** BKP\_PRODUTO**(**

IDBACKUP INT **PRIMARY** **KEY** AUTO\_INCREMENT**,**

IDPRODUTO INT**,**

NOME VARCHAR**(**30**),**

VALOR\_ORIGINAL FLOAT**(**10**,**2**),**

VALOR\_ALTERADO FLOAT**(**10**,**2**),**

**DATA** DATETIME**,**

USUARIO VARCHAR**(**30**),**

EVENTO CHAR**(**1**)**

**);**

2- Criação da trigger de backup:

delimiter $

**create** **trigger** bkp\_produto\_update

**before** **update** **on** produto

**for** **each** **row**

**begin**

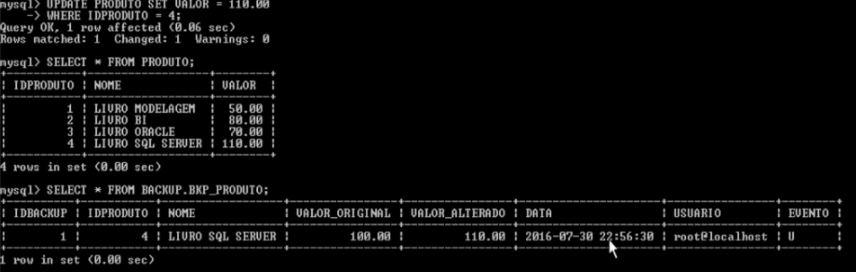
**insert** **into** **backup.**bkp\_produto **values(null,** **old.**idproduto**,** **old.**nome**,** **old.**valor**,** **new.**valor**,** now**(),** **current\_user(),** 'u'**);**

**end**

$

delimiter **;**

3- A tabela de backup (backup.bkp\_produto) após um update na tabela produto:



# Seção 20: Mais modelagem

# Seção 21: Programe, programe!

## Introdução aos cursores

## E mais cursores!

# Seção 22: Normalizando mais!

## 2 e 3 Formas Normais!

## Vamos praticar? Parte 1

## Criando as Constraints

# Seção 23: E o Business Intelligence?

## Bancos Relacionais x Business Intelligence

# Seção 24: Módulos SQL Server – Elevando o nível

## O client do SQL Server e seus Bancos de Sistema

## A cláusula GO

# Seção 25: Arquitetura do SQL Server

## Arquivos LDF e MDF

## Particionando um banco de dados fisicamente

# Seção 26: Vamos aos Códigos!

## Constraints Nomeadas, Identify, SP\_COLUMNS e SP\_HELP

## A função IFNULL(), Clausula ambígua e a função getdate()

## Trabalhando com Datas

## Conversões de tipos de dados

## A função Charindex

## Utilizando Bulk Insert – Desafio utilizando CHARINDEX

## Correção do desafio – Exibindo o Saldo

# Seção 27: Triggers no SQL Server

## Triggers de DML

## Triggers de DML – Parte 2

## Simplificando! Otimize suas triggers utilizando select

## Triggers de Range – Introdução à Transações

# Seção 28: Programando procedures

## Procedures estáticas, dinâmicas, com parâmetros de entrada e de saída!

## Criando procedures como regra de negócio

# Seção 29: Introdução à linguagem TSQL

## Introdução a TSQL – CAST e CONVERT para padrões da datas

## Atribuindo resultados a variáveis

## Trabalhando com estruturas condicionais – IF ELSE

## Estruturas de loop com While

# Seção 30: Módulo PostgreSQL

## Criando o Primeiro Banco de Dados

## Datas – Trabalhando com Datastyle

## Criando o Banco de Dados do Projeto

## Introdução a Funções de Agregapção

## Média

## Principais Medidas Estatísticas

## Concluindo a Análise Estatística

## Modelagem de Banco de Dados x Modelagem para Data Science

## Importando Arquivos e Verificando a MODA

## Amplitude de um Set de Dados

## Desvio Padrão e Variância

## Mediana

## Coeficiente de Variação

## Moda

## Export Formato Colunar

## Arquitetura do Ambiente

## Entendendo a Estrutura e Organização

## Programando a Sincronização

## Sincronizando Registros Deletados

## Exercício – Salários

## Machine Learning – Criando colunas Dummy

## Introdução aos filtros

## Filtros de contadores

## Formatando Strings

# Seção 31: Módulo Oracle Express

## Instalação e criação de conexões no SQL Developer

# Seção 32: Degustação Módulo Oracle 11g – CONTEÚDO TEMPORÁRIO

## Sobre o Módulo Oracle

## Ambiente Geral Oracle

## Estruturas Lógicas

## Tablespaces

## Estruturas Físicas

## Archive e ASM

## Arquivos de Parâmetros

## Trace e PSWD

## Default Tablespace

## Estruturas de Memória – Parte 01

## Entendendo o Redolog

## Estrutura de PGA

## Configurando Processos

## Analisando Queries no Oracle

## Utilitários Linus e Windows

## Criando o Servidor

## Snapshot e configuração de Boot

## Instalando o Oracle Linux

## Mídias do Oracle Linux

## Conexões de Rede

## Conectando-se ao Linux

## Configurando o Linux

## Configurando o Linux – Parte 2

## Configurações Finais

## Criando o primeiro Snapshot

## Mídias do Oracle 11g

## Xming

## Configurando o Xming

## Instalando o SGBD

## Snapshot de SGBD

## Criando o Banco de Dados

## O Enterprise Manager

## O ambiente windows

## Configurando o servidor Windows

## Instalando o SGBD

## Criando o Banco de Dados

## Criando as variáveis

## Instalando as Ferramentas

## Dicionário de Dados

## Tablespace e Tabelas

## Formatando Colunas

## Configurando o SQL Developer

## Customizando Tablespaces

## Sequences

## Alterando Tablespaces

## Pseudo Colunas

## Triggers – Parte 01

## Triggers – Parte 02

## Triggers – Parte 03

## Triggers – Parte 04

## Operações com Views

## Deferrable Constraints – Parte 01

## Deferable – Parte 02

## Deferable – Parte 03

# Seção 33: Introdução ao BIG DATA

## Introdução – parte 01

## Introdução – parte 02

## Introdução – parte 03

## Introdução – parte 04

## Carreiras de Data Science

## Big Data Real Time e Cloud Computing

## O que é Big Data – Parte 01

## O que é Big Data – Parte 02

## O que é Big Data – Parte 03

## O que é Big Data – Parte 04

## O que é Big Data – Parte 05

## Ciclo de Vida de Business Intelligence