

**MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DA EDUCAÇÃO SUPERIOR**

**CENTRO FEDERAL DE EDUCAÇÃO
TECNOLÓGICA
CELSO SUCKOW DA FONSECA
Unidade Descentralizada de Nova Friburgo**

BANCO DE DADOS I

**Professor:
RAFAEL GUIMARÃES RODRIGUES**

ÍNDICE:

1. Considerações iniciais

1. Sistemas de Bancos de Dados
2. Revisão de Modelagem de Dados
 1. Modelos Conceitual, Lógico e Físico
 2. DER (Peter Chen)
 3. Definição de relacionamentos e Cardinalidades
 4. Definição de chaves primárias
 5. Normalização
 6. Outras dicas para uma boa modelagem
 7. Desnormalização
3. Instalação do Mysql
4. SQL e suas subdivisões
5. Criando sua base de dados
6. Criando sua tabela e determinando os tipos das colunas
7. Consultando e alterando a estrutura de sua base de dados, tabelas e colunas
8. Removendo bases, tabelas e colunas
9. Determinando a chave primária (simples ou composta)
10. Tipos de dados
11. Conhecendo as bases de dados default do MySQL

2. Manipulando dados com os comandos da DML

1. Inserindo dados usando o comando INSERT
2. Selecionando registros com o comando SELECT
3. Ordenando os resultados com ORDER BY
4. Limitando as linhas através de LIMIT
5. Atualizando registros com o comando UPDATE
6. Removendo registros com o comando DELETE
7. Renomeando colunas selecionadas usando alias
8. Filtro de registros com a clausula WHERE
9. Funções usadas em consultas (DISTINCT, CONCAT, SUBSTRING etc)
10. Operadores BETWEEN e IN para facilitar os filtros de registros
11. Lista de Exercícios nº 1
12. O operador LIKE
13. Lista de Exercícios nº 2
14. Exercícios de reestruturação da base vendas, com o Professor.

3. Juntando várias tabelas

1. Juntando tabelas de forma simples
2. Juntando tabelas com INNER JOIN e OUTER JOIN
3. Juntando consultas com UNION
4. Lista de Exercícios nº 3
5. Comandos da DDL – Continuação...
6. Comandos DML – Continuação.... INSERT através de um SELECT. Fazendo ajustes em nossa base de dados.
7. Funções Matemáticas

8. Atualizando registros com o comando UPDATE
 9. Removendo registros com o comando DELETE
 10. Renomeando colunas selecionadas usando alias
 11. Filtro de registros com a cláusula WHERE
 12. Funções usadas em consultas (DISTINCT, CONCAT, SUBSTRING etc)
 13. Operadores BETWEEN e IN para facilitar os filtros de registros
 14. Lista de Exercícios nº 1
 15. O operador LIKE
-
4. **Agrupando dados e fazendo consultas mais inteligentes**
 1. Funções de Agregação
 2. Usando GROUP BY para agrupar registros de acordo com as funções de agregação
 3. Utilizando a cláusula HAVING
 4. Realizando consultas com JOIN, GROUP BY, Funções Matemáticas (de agregação) e HAVING
 5. **Integridade Referencial**
 1. Criando tabelas com regras de integridade;
 2. Alterando tabelas existentes acrescentando regras de integridade.
 3. Regras para apagar e atualizar registros com relacionamentos.
 6. **Controle de transações**
 1. Conceito de transação;
 2. Fazendo o MySQL trabalhar em modo de transação.
 3. Confirmando ou cancelando uma transação.
 7. **Subconsultas**
 8. **Funções para trabalhar com datas**
 9. **Funções numéricas**
 10. **Funções para trabalhar com strings**
 11. **Exercícios diversos**

❖ 1 – Considerações Iniciais

Tecnologias de Bancos de Dados conhecidas:

- HIERÁRQUICO
- ORIENTADO A OBJETOS (não “pegou”)
- RELACIONAL (predominam atualmente)
- OBJETO-RELACIONAL
- GEOGRÁFICOS
- NOSQL

SGBDs Relacionais:

- **Arquitetura CLIENTE / SERVIDOR:** ORACLE, DB2, SQL SERVER, INTERBASE, FIREBIRD, POSTGRES, MYSQL etc.

Sistema escolhido para as disciplinas:

→SGBD (MySql)

Histórico dos SGBDs

1960 (final) – O matemático **Edgar Codd** publicou o artigo “**A Relational Model of Data for Large Shared Data Banks**” com base na Álgebra Relacional.

1973 – A **IBM** iniciou o seu 1º SGBDR, o **System R**. Este projeto provou a viabilidade do modelo matemático de Codd. Foi o berço do SQL.

1974 – Pesquisadores da **IBM** propuseram uma linguagem declarativa para expressar as operações realizadas no modelo relacional. Inicialmente foi chamada de **SEQUEL** (Structured English QUery Language), que por motivos legais, a sigla foi alterada para **SQL**.

1979 – A **Oracle** lança comercialmente o primeiro SGBDR.

1981 – A **IBM** lança comercialmente o SQL/DS.

1983 – A **IBM** lança comercialmente o DB2.

1986 – A ANSI publicou a 1ª versão do padrão **SQL**, que ficou conhecido como SQL-86.

1987 – O padrão foi publicado como uma norma internacional pela ISO (International organization for Standardization).

1989 – Foi publicada uma nova padronização pela ISO incluindo recursos como integridade referencial, valores null e default, check constraints e outros.

2024 – O modelo relacional continua dominando o mercado ainda que novas tecnologias como NoSQL tenham surgido.

Características de um SGBD

CONTROLE DE REDUNDÂNCIA: A redundância, ou seja, a repetição de dados, deve ser evitada para se minimizar possibilidade de inconsistências.

COMPARTILHAMENTO DE DADOS: Em um ambiente multiusuários deve-se possibilitar a manipulação simultânea de dados distintos ou dos mesmos dados conforme regras abaixo.

→ **CONTROLE DE ACESSO:** Verificação automática do tipo de acesso pedido por cada usuário. Os níveis de segurança são estabelecidos para cada usuário independentemente de acordo com suas necessidades. A identificação de cada usuário, por parte do SGBD, é feita pelo nome e senha cadastrados.

→ **CONTROLE DE TRANSAÇÃO:** Transação é o conjunto de operações que devem ser executadas completamente.

São normalmente usadas em situações críticas (atualizações ou inclusões) de longa duração que podem afetar a consistência do BD. O SGBD deve utilizar mecanismos internos para que nenhuma falha ocorra durante a execução da transação.

→ **ACESSO EM MÚLTIPLAS INTERFACES:** Possibilidade de usar diversas interfaces mesmo se o SGBD estiver sendo utilizado. Por exemplo: Se existe uma aplicação em Delphi com o SGBD Interbase, se trocar a linguagem para Visual Basic, não é necessário fazer alterações no SGBD.

→ **RESTRIÇÕES DE INTEGRIDADE:** Estabelecimento de um formato para os dados inseridos de modo a garantir uma certa integridade e facilitar o armazenamento. Algumas regras de integridade são estabelecidas pelo próprio SGBD para manter o BD consistente e outras são definidas pelo DBA por meio de sentenças condicionais que são verificadas toda vez que um dado é armazenado no BD.

BACKUP E RECUPERAÇÃO: Estabelecer o backup automático do BD total ou parcial em momentos estabelecidos pelo DBA. Proporcionar proteção contra a perda de informações devido a falhas no dispositivo de armazenamento (discos).

INDEPENDÊNCIA DE DADOS: A descrição física dos arquivos é mantida internamente pelo SGBD e é de sua inteira responsabilidade e exclusividade. Programas aplicativos não dispõem da descrição física e sim de uma descrição externa.

→ **INDEXAÇÃO AUTOMÁTICA:** Com a indicação explícita dos atributos que serão mais utilizados em consultas, o SGBD cria os arquivos de indexação que tornarão mais rápidas as pesquisas. A estrutura de indexação e de organização dos arquivos de dados é próprio de cada SGBD e normalmente não é de domínio dos usuários comuns.

VANTAGENS:

- Sistema sofisticado.
- Arquitetura Cliente / Servidor
- Integridade referencial
- Nível de Segurança altíssimo
- Apenas a informação solicitada trafega pela rede e não a tabela inteira.
- Controle de usuários eficaz e confiável.
- Controles, rotinas, regras e restrições podem ocorrer diretamente na base de dados evitando tráfego desnecessário pela rede, melhorando a performance e aliviando o processamento nas máquinas cliente.
- Diminui consideravelmente os requisitos de memória nas máquinas cliente.
- Boa parte dos SGBD's são OPEN SOURCE (Código Aberto) e de livre distribuição.
- Estão em constante evolução.

○ **Quando usar um SGBD relacional?**

- Sistemas que funcionem em redes locais, web ou ambos.
- Sistemas híbridos (Delphi e PHP por exemplo)
- Sistemas que necessitem de regras rígidas de segurança e alta confiabilidade.

❖ 4 - Conhecendo os SGBD'S e justificando a escolha pelo MySQL

→ ORACLE: 1979 – A Oracle lança comercialmente o primeiro SGBDR.

- O melhor e mais sofisticado Banco de dados do Mundo.
- Multiplataforma.
- Seria covardia compará-lo aos demais SGBD's.
- O ORACLE é mais que um SGBD, é quase uma plataforma de desenvolvimento (SOFTWARE e BANCO DE DADOS ao mesmo tempo)
- Possui diversas Tecnologias de acesso a dados, dentre elas (BDE – ODBC, Zeos, entre outras)
 - **Por que não foi escolhido?**
 - Código Fechado.
 - Licença caríssima, totalmente fora da nossa realidade, inclusive em se tratando de mercado de trabalho.
 - Alto nível de complexidade e, devido a seu custo elevado, pouco difundido.
 - Consequentemente possui poucas referências gratuitas para pesquisa e desenvolvimento.
 - Devido ao seu custo elevado, é pouco difundido.

→ DB2: 1983 – A IBM lança comercialmente o DB2.

- Não é preciso entrar em detalhes. O DB2 seria o ORACLE da IBM.
- “Disputa” com a ORACLE, perdendo é claro, o mercado de MAINFRAMES.

→ SQL SERVER:

- Excelente Banco de Dados.
- Ótimas Ferramentas de Administração: Query Analyzer, Enterprise Manager, Profiler, entre outras.
- Possui diversas Tecnologias de acesso, dentre elas (BDE – ODBC, ADO, Zeos, entre outras)
 - **Por que não foi escolhido?**
 - Código Fechado pertencente à Microsoft.
 - Licença muito cara, inclusive em se tratando de mercado de trabalho.
 - Banco de Dados pesado se comparado aos Open Source.

→ INTERBASE:

- Bom Banco de Dados.
- Por ser uma ferramenta da Borland, se integra muito bem com o Delphi e a BDE.
- Tecnologias de Acesso a dados: BDE , DBExpress, IBX (ambos da Borland)
 - **Por que não foi escolhido?**
 - Código Fechado pertencente à Borland.
 - Esteve Estagnado por alguns anos, assim como a BDE.

- Perdeu espaço para o Firebird.

→ FIREBIRD:

- Excelente Banco de Dados.
- Multiplataforma.
- Tecnologias de Acesso a dados: DBExpress, IBEXPERT, Zeos, entre outras.
- A partir da Versão 6 do Delphi a Borland resolveu abrir o código do Interbase e foi aí que paralelamente um grupo de desenvolvedores resolveu criar o Firebird.
- Conclusão: Na versão 7 do Delphi a Borland fechou novamente o Interbase, que ficou estagnado por um tempo, ao contrário do Firebird.
- Continua em constante desenvolvimento, inclusive já tendo superado o Interbase que foi o seu precursor.
- OPEN SOURCE (Código Aberto)
- Distribuição Gratuita.
- Bastante difundido.
- Estável e Confiável.
 - **Por que não foi escolhido?**
 - Poderia perfeitamente ser o objeto de nosso estudo. Uma pequena desvantagem em relação ao Mysql seria o fato de ainda não estar suficientemente difundido para Web.
 - O MySql está no Mercado há mais tempo e é muito mais utilizado, conseqüentemente, mais testado.

→ POSTGRES:

- Excelente Banco de Dados.
- Multiplataforma.
- Tecnologias de acesso a dados: PsqLODBC, Open Link ODBC, Zeos, entre outras.
- Está em constante desenvolvimento.
- OPEN SOURCE (Código Aberto)
- Distribuição Gratuita.
- Bastante difundido, inclusive na Web.
- Estável e Confiável.
- Bastante Robusto. Possui recursos como Triggers , Stored Procedures e Views desde suas primeiras versões, fato que o tornava superior ao Mysql.
 - **Por que não foi escolhido?**
 - Assim como o Firebird, poderia perfeitamente ser o objeto de nosso estudo.
 - O MySql está no Mercado há mais tempo e é muito mais utilizado, conseqüentemente, há mais referências.
 - O MySql tem arquitetura mais simples, mais leve e inegavelmente MAIS RÁPIDO.
 - Didaticamente o MySql é mais aconselhável por ser um SGBD de fácil compreensão. Diria que é mais AMIGÁVEL.
 - Em plataforma Windows o MySQL está mais “calejado”.

→ MYSQL:

- Excelente Banco de Dados.
- Multiplataforma.
- Tecnologias de acesso a dados: BDE-MYODBC, ADO, Zeos, entre outras
- Está em Constante desenvolvimento.
- OPEN SOURCE (Código Aberto)
- Distribuição Gratuita (em parte).
- Estável e Confiável.
- Possui TODAS as demais características do Firebird e do Postgres.
 - **Por que foi escolhido?**
 - É reconhecidamente o SGBD mais leve e mais rápido. Afinal, inicialmente foi feito para a Web.
 - Está no mercado há muito mais tempo do que o Firebird e o Postgres. Portanto, mais testado e mais confiável.
 - Suas desvantagens em relação ao Postgres foram supridas a partir das versões 4 e 5.
 - Didaticamente falando é o SGBD de mais fácil compreensão, por sua arquitetura simples.
 - Em relação ao Postgres está sendo usado há muito mais tempo em plataforma Windows.

Para não restarem dúvidas:

- O Mysql é, SIMPLEMENTE, o **Banco de Dados mais utilizado no mundo!!!!**
- É a escolha de grandes Empresas, dentre elas:
 - NASA, SONY, MOTOROLA, HP, TELEMAR, BRADESCO
 - NOKIA, SUZUKI, U. S ARMY, U. S NAVY
- O logotipo do MySQL é um golfinho.
- “A escolha revela uma espécie de mascote, traduzida em um animal inteligente, rápido, livre de gordura/excessos e que desbrava com facilidade os oceanos (de dados) como o próprio MySQL”.

Conectando-se a uma base de dados via prompt:

No Terminal Unix ou prompt do MS-DOS digite:

mysql -h nome_do_servidor -u nome_do_usuario -p
e pressione Enter.

Explicando esta linha acima temos que:

mysql é o comando que vai chamar o Cliente Mysql para administração,
-h nome_do_servidor, especifica o servidor a ser utilizado,

-u nome_do_usuario, especifica o usuário que tem privilegio para acessar o mysql e
-p a chamada para a senha do cliente mysql.

Após pressionar Enter, uma linha solicitando a senha do MySQL aparece, insira a senha e pressione enter.

Pronto! Você já está conectado ao MySQL.

Desconectando-se...

Para se desconectar do MySQL é mais facil ainda.

Basta você digitar:

quit

e pressionar enter, e pronto!

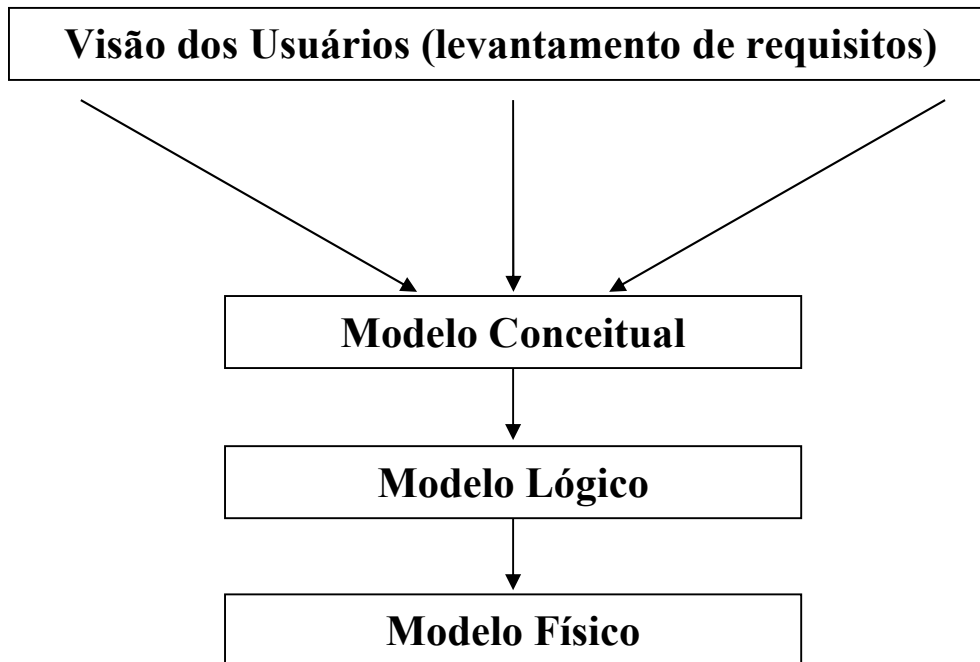
Conhecendo as bases de dados default do MYSQL:

- **Information_Schema** (Informações gerais sobre bases de dados, tabelas, campos e seus tipos, rotinas, procedimentos e mais uma infinidade de informações sobre os seu banco de dados. Entenderemos melhor esta base mais adiante.)
- **Mysql** (É a principal base de dados do seu Banco. Onde ficam armazenados seus usuários, permissões e até mesmo tópicos de ajuda sobre os comandos da linguagem SQL)
- **Test** (É uma base de dados default para que o usuário comece a fazer suas simulações)

Modelagem de Dados

Modelo de Dados:

→ Representação do mundo real(minimundo) baseada em um conjunto de entidades e seus relacionamentos.



→ Modelo Conceitual

- Nesta etapa a participação do usuário é ESSENCIAL.
- Nesse nível não existem limitações tecnológicas.
- Neste modelo temos:
 - Visão geral do negócio;
 - Linguagem comum a técnicos e usuários;
 - Possui somente as entidades e atributos principais;
 - Pode conter relacionamentos n:m
 - DER (Diagrama Entidade Relacionamento)

→ Modelo Lógico

- Deve-se considerar a Tecnologia a ser utilizada (Relacional, Hierárquico, OO, NoSQL etc.)
- Deriva do Modelo Conceitual
- Neste modelo temos:
 - Entidades associativas no lugar de relacionamentos com atributos ou relacionamentos n:m;
 - Definição das chaves primárias das entidades;
 - Normalização até a 3FN;
 - Padrão de nomenclatura bem definido;

→ Modelo Físico

- Deve-se definir não apenas a Tecnologia, mas também o SGBD a ser utilizado, considerando suas características e limitações.
- Deriva do Modelo Lógico.
- Neste modelo temos:
 - Transformação de Entidades em Tabelas e atributos em colunas ou campos.
 - Criação das tabelas em determinado SGBD;

- Definição dos índices visando performance;
- Definição de regras de negócio, restrições, SP, Triggers, UDF e UDT.

Diagrama de Entidade e Relacionamento (DER)

→ Proposto por Peter Chen no início da década de 70.

→ Entidades

- Representação abstrata de um objeto do mundo real.
- Geralmente representada por um substantivo (Carro, Pessoa, Fornecedor, Cliente etc.).

→ Relacionamentos

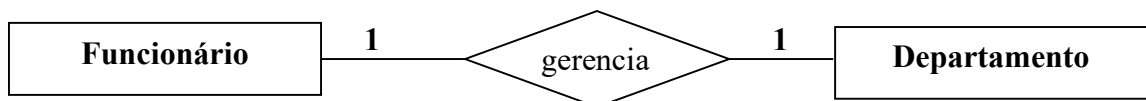
- Entidades são conectadas umas às outras através de relacionamentos que possuem determinada cardinalidade (1:1, 1:n, n:n)
- Em geral, relacionamentos são representados por verbos (associa, contém, gerencia, efetua, etc.).

→ Cardinalidade de um Relacionamento

Indica o número de instâncias de entidades que podem estar envolvidas umas com as outras através de um relacionamento.

→ Cardinalidade um-para-um

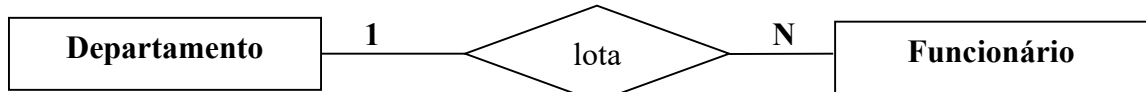
- Uma Entidade A está associada a no máximo uma entidade B e uma entidade B está associada a no máximo 1 entidade A..



Obs: Chave estrangeira em uma das entidades.

→ Cardinalidade um-para-muitos

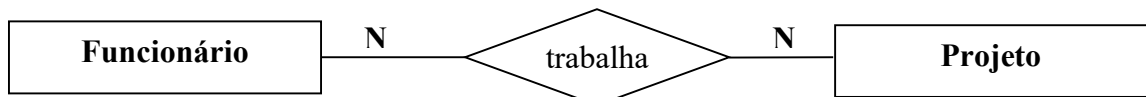
- Uma Entidade A está associada a uma ou muitas entidades B e uma entidade B está associada a no máximo 1 entidade A..



Obs: Chave estrangeira na direção de muitos.

→ Cardinalidade muitos-para-muitos

- Uma Entidade A está associada a uma ou muitas entidades B e uma entidade B está associada a no máximo 1 entidade A..



Obs: Relacionamentos n:n geram nova tabela.

→ Atributos (campos)

- Elemento de dado que contém o valor de uma propriedade (característica) de uma entidade.

Classificação dos Atributos:

→ Atributo Simples

Não tem outros atributos aninhados, apenas o valor. Ex: Nome.

→ Atributo Composto

É composto por outros atributos. Ex: Endereço.

→ Atributo Monovalorado

Um único valor para cada instância. Ex: Nome.

→ Atributo Multivalorado

Mais de um valor para cada entidade. Ex: Telefone, Dependentes.

→ Atributo Derivado

Seu valor é calculado ou obtido a partir de outro atributo. Ex: Idade (pode ser obtido a partir da data_nascimento), Preço_Venda (pode ser obtido a partir de preço_custo).

→ Atributo Chave

Identifica unicamente cada entidade. Ex: funcionário_id, CPF etc.

Chave (Definição):

É o conjunto de um ou mais atributos que identificam unicamente cada entidade (registro) no conjunto-entidade (tabela).

Em se tratando de chave é importante saber que:

- Uma chave não pode conter valor nulo.

→ Chave Simples

Quando um único atributo é suficiente para garantir a unicidade das tuplas.

Uma chave simples pode ser:

- **Chave Natural** → Quando um atributo do mundo real for suficiente para garantir a unicidade. Ex: Cpf, CNPJ, Matrícula etc.
- **Chave Artificial** → Quando não há um atributo do mundo real capaz de garantir a unicidade. Nesse caso usamos uma chave artificial. Geralmente trata-se de valores sequenciais (auto incrementável).

→ Chave Composta

Quando um único atributo não é suficiente para garantir a unicidade das tuplas. Nesse caso formamos a chave com dois ou mais atributos. A combinação dos valores deve garantir a unicidade.

→ Entidade Fraca

Não possui identidade própria. Sua chave primária é composta pela chave estrangeira proveniente da entidade “dona” concatenada a um identificador de si própria (pode repetir para diferentes instâncias da entidade dona).



→ Normalização (Modelo Lógico)

É o conjunto de regras que visa eliminar redundâncias e minimizar as anomalias de modificação dos dados dando maior flexibilidade em sua utilização.

→ Formas Normais

Hoje em dia podemos considerar que existem 6 formas normais. No entanto o usual é normalizar até a 3FN. As outras 3 formas tratam de casos muito específicos e de conteúdo predominantemente teórico. São elas a Forma de Boyce Codd, conhecida como BCFN, 4FN e 5FN.

→ 1a Forma Normal (1FN)

Eliminar subgrupos de atributos repetitivos (combinações repetidas), atributos compostos e atributos multivalorados.

Exemplo:

cliente = {cpfCliente, nomeCliente, endereçoCliente, dependenteCliente}

A relação acima não está na 1FN porque Endereço é composto e dependente é multivalorado.

Aplicando a 1FN:

cliente = {cpfCliente, nomeCliente, rua, numero, bairro, cidade, uf}

dependente = {cpfCliente, nomeDependente, dataNascimento}

Note que o atributo composto endereçoCliente foi decomposto.

Outro Exemplo:

vendas								
nNota	dataNota	codCliente	nomeCliente	codProduto	nomeProduto	qtde	vUnit	vTotal
12345	10/01/2009	19	Fulano	120	Laranja	12	R\$2,40	R\$2,40
12345	10/01/2009	19	Fulano	115	Maçã	5	R\$2,00	R\$2,00
12345	10/01/2009	19	Fulano	200	Leite	3	R\$4,50	R\$4,50
22222	12/01/2009	27	Beltrano	400	Sabonete	10	R\$10,00	R\$10,00
22222	12/01/2009	27	Beltrano	128	Escova	1	R\$6,00	R\$6,00

A relação acima não está na 1FN porque contém subgrupos de atributos repetitivos (SubGrupo nNota, dataNota, codCliente e nomeCliente). Repare que a combinação se repete para cada produto de uma mesma venda.

Aplicando a 1FN:

nota_Fiscal			
nNota	dataNota	codCliente	nomeCliente
12345	10/01/2009	19	Fulano
22222	12/01/2009	27	Beltrano

produtos_Vendidos					
nNota	codProduto	nomeProduto	qtde	vUnit	vTotal
12345	120	Laranja	12	R\$0,20	R\$2,40
12345	115	Maçã	5	R\$0,40	R\$2,00
12345	200	Leite	3	R\$1,50	R\$4,50
22222	400	Sabonete	10	R\$1,00	R\$10,00
22222	128	Escova	1	R\$6,00	R\$6,00

Repare que decompomos a entidade em duas e na segunda entidade só repetimos o campo chave (nNota).

→ 2a Forma Normal (2FN):

1FN + Os atributos não-chave devem depender funcionalmente da totalidade da chave primária e não somente de parte dela.

Ou seja, não pode haver dependência funcional parcial!

Sendo assim, a 2FN só se aplica às tabelas com chave primária composta!

Exemplo:

disciplinasCursadas = {matricula,codCurso,codDisciplina, nomeAluno,notaFinal}

A relação acima não está na 2FN porque o atributo não-chave nomeAluno depende funcionalmente de apenas parte da chave (matricula).

Aplicando a 2FN:

aluno = {matricula, nomeAluno}

disciplinasCursadas = {matricula,codCurso,codDisciplina,notaFinal }

Nestes casos, cria-se uma tabela nova com parte da chave e seus atributos não-chave dependentes funcionalmente.

Outro Exemplo:

produtosVendidos					
nNota	codProduto	nomeProduto	qtde	vUnit	vTotal
12345	120	Laranja	12	R\$0,20	R\$2,40
12345	115	Maçã	5	R\$0,40	R\$2,00
12345	200	Leite	3	R\$1,50	R\$4,50
22222	400	Sabonete	10	R\$1,00	R\$10,00
22222	128	Escova	1	R\$6,00	R\$6,00

A relação acima não está na 2FN porque o atributo não-chave produto depende funcionalmente de apenas parte da chave primária (codProduto).

Aplicando a 2FN:

produtos		
codProduto	nomeProduto	valor
115	Maçã	R\$0,45
120	Laranja	R\$0,22
128	Escova	R\$6,00
200	Leite	R\$1,60
400	Sabonete	R\$1,10

produtosVendidos				
nNota	codProduto	qtde	vUnit	vTotal
12345	120	12	R\$2,40	R\$2,40
12345	115	5	R\$2,00	R\$2,00
12345	200	3	R\$4,50	R\$4,50
22222	400	10	R\$10,00	R\$10,00
22222	128	1	R\$6,00	R\$6,00

Repare que decompomos a entidade em duas e na segunda entidade só repetimos o campo chave (Num_Nota).

→ 3a Forma Normal (3FN):

2FN + Os atributos não-chave não pode depender de outro atributo não-chave.
 Ou seja, não pode haver dependência transitiva!

Exemplo:

notaFiscal			
nNota	dataNota	codCliente	nomeCliente
12345	10/01/2009	19	Fulano
22222	12/01/2009	27	Beltrano

A relação acima não está na 3FN porque o atributo não-chave nomeCliente depende funcionalmente do atributo não-chave codCliente.

Aplicando a 3FN:

Clientes	
codCliente	nomeCliente
19	Fulano
27	Beltrano

notaFiscal		
nNota	dataNota	codCliente
12345	10/01/2009	19
22222	12/01/2009	27

Os atributos que caracterizam a dependência transitiva vão para uma outra tabela e um dos atributos vira chave.

→Dicas de Modelagem (Modelo Conceitual)

1. Elimine qualquer redundância de dados. Com exceção das chaves estrangeiras (que referenciam chaves primárias em outras tabelas) cada informação deve estar em uma única tabela. Ex: O nome do Cliente só precisa estar na tabela “cliente”. Não precisa estar na tabela “pedido”.
2. Adote um padrão para dar nome a entidades, atributos e relacionamentos. Ex: Nomes de entidades no singular e com a primeira letra minúscula e padrão snake_case.
3. Dê nomes significativos a entidades, atributos e relacionamentos. Isso facilita a compreensão do modelo.
4. Relacionamentos N:M ou relacionamentos que possuem atributos geralmente se transformam em tabela associativa no modelo lógico. O nome do relacionamento pode ser usado como nome da nova tabela.
5. Ao dar nomes a atributos que são chave estrangeira, procure sempre referenciar o nome da entidade. Ex: use cliente_id (nome da tabela_nome da chave). Isso ajuda a identificar a origem.

→Dicas de Modelagem (Modelo Lógico)

1. Toda entidade do Modelo Conceitual vira uma tabela no Modelo Lógico.
2. Elimine colunas repetidas no seu modelo. Se uma coluna se repete em várias tabelas é provável que haja colunas desnecessárias.
3. Campos de texto com tamanho variável ocupam menos espaço.
4. Na hora de definir a chave de sua tabela, prefira atributos numéricos.
5. Evite definir uma chave composta por muitos atributos. Isso complica a programação.
6. Se possível, elimine atributos derivados ou calculados. Estas informações podem ser obtidas através de simples consultas.
7. Toda tabela deve ter uma chave primária!!
8. Definir chaves naturais (CPF, CNPJ, Matrícula, RG etc.) é legal, mas uma chave artificial (campo sequencial) sempre cai bem. Use índices únicos para os campos candidatos a chaves naturais.
9. Chaves estrangeiras devem corresponder ao valor de uma chave primária em outra tabela associada ou conter valor nulo quando não se tratar de campo obrigatório.
10. Evite criar tabelas com excesso de campos. Isso interfere no desempenho. Use as formas normais para distribuir os campos. Evite apenas os excessos. Normalizar em excesso também atrapalha.
11. Em relacionamentos 1:1 uma das entidades deve conter uma chave estrangeira. Geralmente a entidade mais acessada.
12. Em relacionamentos 1:N a chave estrangeira vai para o lado N.
13. Relacionamentos N:M geram nova tabela com relacionamentos 1:N nas pontas.

Ex: aluno (N) \leftrightarrow (N) disciplinas

aluno (1) \rightarrow (N) aluno_disciplina (N) \leftarrow 1 disciplina

14. Desnormalizar pode ser útil para ganhar desempenho eliminando os custos das junções. Isso só é válido em casos muito específicos.

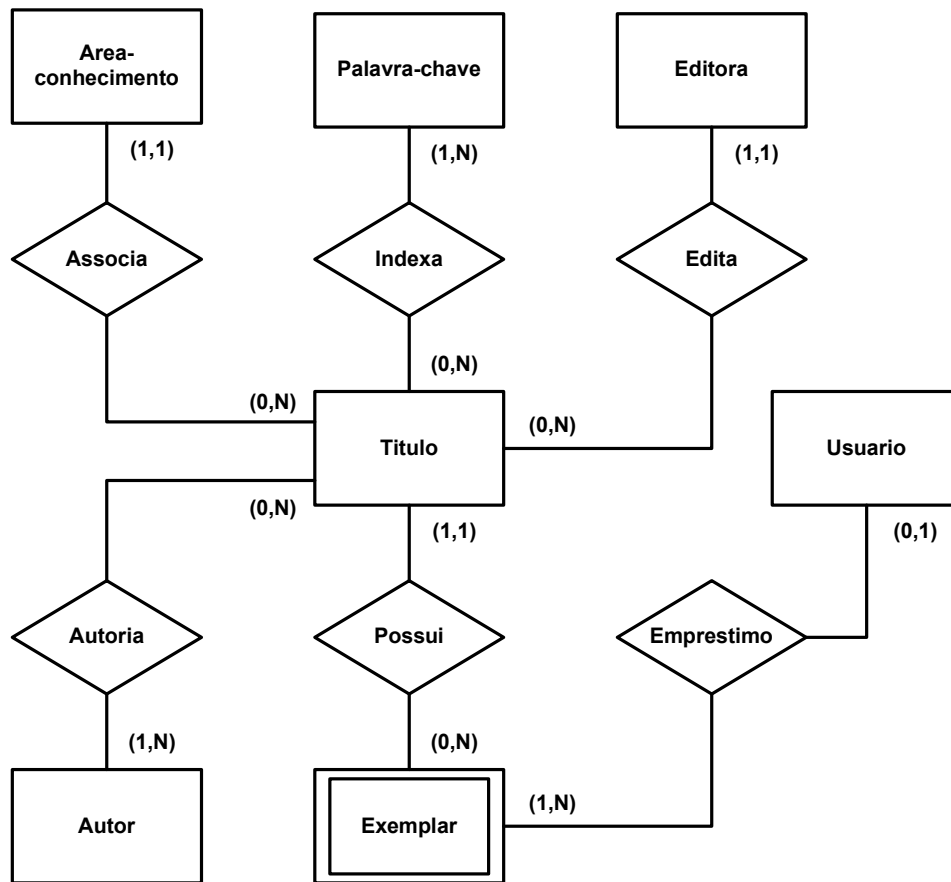
→Dicas de Modelagem (Modelo Físico)

1. A criação de índices para chaves estrangeiras tende a melhorar o desempenho das consultas complexas ou que envolvam muitos registros.
2. Mesmo quando houver uma chave natural, não complique. Use uma chave artificial.

Exercício de Modelagem: Construir os Modelos Conceitual, Lógico e Físico de um Sistema de controle e gerenciamento de empréstimos de livros de uma biblioteca acadêmica.

→Declaração de Escopo (Biblioteca):

- A biblioteca dispõe de livros, também denominados títulos. Estes possuem nome, autores e editoras. Cada título pertence a uma área de conhecimento e possui um código único de identificação.
- Cada título possui vários exemplares. Cada exemplar possui um código único de identificação.
- Cada título pode ter vários autores e um mesmo autor pode ter escrito vários títulos. Um autor possui código, nome, telefone e endereço.
- As editoras possuem código, nome, telefone e endereço.
- As áreas de conhecimento possuem código e uma descrição.
- Usuários, que podem ser alunos, professores ou funcionários, tomam livros emprestados por uma semana. A data de empréstimo é importante no processo.
- Cada usuário possui um código, nome, telefone e endereço.
- Cada título possui várias palavras-chave e uma palavra-chave pode estar ligada a vários títulos. Uma palavra-chave possui código e descrição.



Area-conhecimento (Codigo, Descricao)

Palavra-chave (Codigo, Descricao)

→ Editora (Codigo, Nome, Telefone, Endereco)

Titulo (Codigo, Nome)

Autor (Codigo, Nome, Telefone, Endereco)

Exemplar (Codigo)

Usuario (Codigo, Nome, Telefone, Endereco, Tipo)

Emprestimo (Data)

BANCO DE DADOS

❖ 3 – SQL – (Structured Query Language) Linguagem de Consulta Estruturada

→ Esta linguagem é universal, principalmente em termos de Bancos de Dados Relacionais.

→ As diferenças de sintaxe, principalmente entre os SGBD's, são muito sutis.

Divide-se em 3 ou até 4 categorias, dependendo do Sistema de Banco de dados usado ou a literatura de referência. São elas:

- **DML (Linguagem de Manipulação de Dados):**
 - Consultar, Alterar, Incluir e Excluir dados.
 - Comandos Básicos: SELECT, INSERT, UPDATE, DELETE
- **DDL (Linguagem de Definição de Dados):**
 - Criação e Exclusão de Tabelas e alterações em sua estrutura.
 - Comandos Básicos: CREATE, ALTER, DROP
- **DCL (Linguagem de Controle de Dados):**
 - Controle de Usuários e níveis de privilégio.
 - Comandos Básicos: GRANT e REVOKE.
- **TC (Controle de Transações):**
 - Comandos para colocar as transações em bloco e especificar o início e o fim.
 - Comandos Básicos: COMMIT (ou se confirma todo o bloco de instruções)
 - ROLLBACK (ou não se faz nada, ou seja, volta-se o banco ao estado original, antes do início da transação)

→ DDL (Linguagem de DEFINIÇÃO de Dados)

→ Criando um Banco de Dados

```
CREATE DATABASE <nome_do_banco>;
```

→ Visualizando os Bancos de Dados

```
SHOW DATABASES;
```

→ Ativando um Banco de Dados

```
USE <nome_do_banco>;
```

→ Excluindo um Banco de Dados

```
DROP DATABASE <nome_do_banco>;
```

IMPORTANTE: Todas as tabelas (com ou sem registros) contidas no banco de dados em questão serão apagadas de forma irreversível.

TABELAS

→ Criando uma tabela

```
CREATE [TEMPORARY] TABLE <nome_da_tabela> (<colunas>) [<opções avançadas>];
```

[] = Parâmetros opcionais.

TEMPORARY → Primeiro parâmetro opcional que permite definir se a tabela é apenas temporária. Caso seja definida como TEMPORÁRIA, a mesma só vai existir enquanto durar a conexão do usuário em questão. Caso haja uma queda do banco, ou logoff do usuário, perde-se a tabela e os registros contidos nela.

Recomenda-se usar tabelas temporárias apenas para fins muito específicos.

Colunas:

Cada coluna da tabela deve seguir o seguinte modelo:

```
<nome_da_coluna> <tipo de dado> [NULL | NOT NULL] [DEFAULT <valor>]  
[Auto_INCREMENT] [PRIMARY KEY | INDEX]
```

IMPORTANTE: Não se deve usar espaço para definir nomes de tabelas, bancos de dados ou colunas.

<tipo de dado> → define qual é o tipo de dado contido na coluna. Veremos os tipos possíveis a seguir.

[NULL | NOT NULL] → Define se a coluna é de preenchimento obrigatório (NOT NULL) ou se pode conter valores nulos. Caso a informação seja omitida, a definição padrão é NULL.

[DEFAULT <valor>] → Pode-se definir um valor padrão para a coluna.

[AUTO_INCREMENT] → Define chave artificial auto_incrementável. Se não for o caso, basta omitir a informação.

[PRIMARY KEY | INDEX] → define a coluna em questão como chave primária ou índice. Se não for o caso, basta omitir a informação.

Outra forma de definir colunas como chave primária ou índice é colocar, antes de fechar o parêntese, a seguinte informação: PRIMARY KEY(<nome da coluna>) se a chave for simples ou PRIMARY KEY(<nome_da_coluna1>,<nome_da_coluna2>, ...) se a chave for composta.

INDEX(<nome da coluna>) se o índice for simples ou

INDEX(<nome_da_coluna1>,<nome_da_coluna2>, ...) se o índice for composto.

Falaremos sobre índices mais adiante.

[<opções avançadas>] → Podem se dividir em (separadas por vírgula):

ENGINE=<valor> → Determina qual o mecanismo de armazenamento (tipo de tabela) utilizado. O MySQL dispõe de 9 tipos diferentes. Contudo os mais utilizados são o MyISAM quando a demanda for por velocidade e INNODB quando a demanda for por regras de integridade. Se esta informação for omitida, a tabela será criada usando o mecanismo padrão (INNODB).

AUTO_INCREMENT = <valor> → Define a partir de que valor deve ocorrer a incrementação. Se a informação for omitida, o valor assumido é 1.

Um Exemplo de criação de tabela bem completo:

```
CREATE TABLE Pessoa(  
codPessoa INT NOT NULL AUTO_INCREMENT,  
nomePessoa VARCHAR(60) NOT NULL,  
salario DOUBLE NOT NULL DEFAULT 465,  
PRIMARY KEY(codPessoa),INDEX(nomePessoa))  
ENGINE=INNODB          CHARSET=utf8          COLLATE=utf8_unicode_ci  
AUTO_INCREMENT=10;
```

→ Se você desejar que todas as suas tabelas tenham uma engine específica e critério padrão para caracteres e acentuação, você pode economizar linhas definindo isso na criação da base de dados.

```
CREATE DATABASE minha_base ENGINE=INNODB CHARSET=utf8  
COLLATE=utf8_unicode_ci;
```

→ Renomeando uma tabela

```
RENAME TABLE <nome_da_tabela> TO <novo_nome_da_tabela>;
```

→ Visualizando tabelas

```
SHOW TABLES;
```

→ Visualizando a estrutura de uma tabela

```
DESCRIBE <nome_da_tabela>;
```

Ou

```
SHOW COLUMNS FROM <nome_da_tabela>;
```

→ Alterando uma Tabela

```
ALTER TABLE <nome> <opções>;
```

- **Adicionando uma coluna em uma tabela:**

ALTER TABLE <nome_da_tabela> ADD COLUMN <nome_da_coluna> <definições da coluna> [FIRST | AFTER <coluna_existente>]

EX:

ALTER TABLE pessoa ADD COLUMN telefone VARCHAR(13) NOT NULL AFTER nome_pessoa;

No caso acima, estamos criando a coluna telefone depois (AFTER) da coluna nome_pessoa.

- **Alterando uma coluna em uma tabela:**

ALTER TABLE <nome_da_tabela> CHANGE COLUMN <nome_da_coluna> <novo_nome_da_coluna> <definições da coluna>

Ex:

ALTER TABLE pessoa CHANGE COLUMN telefone telefone_pessoa VARCHAR(13) NOT NULL;

- **Removendo uma coluna de uma tabela:**

ALTER TABLE <nome_da_tabela> DROP COLUMN <nome_da_coluna>;

Ex:

ALTER TABLE Pessoa DROP COLUMN telefone_pessoa;

→ **Alterando o mecanismo de armazenamento de uma tabela:**

ALTER TABLE <nome_da_tabela> ENGINE=<nome_do_mecanismo>;

Ex:

ALTER TABLE pessoa ENGINE=MyISAM;

→ **Excluindo uma tabela**

DROP TABLE <nome_da_tabela>;

Ex:

DROP TABLE pessoa;

IMPORTANTE: Se você deseja excluir somente os registros da tabela, deve usar o comando TRUNCATE TABLE <nome_da_tabela>.

Tipos de dados de MySQL

Lista e descrição dos diferentes tipos de dados de MySQL.

Depois da fase de design da base de dados, e uma vez que se passou a tabelas, é necessário criar as tabelas correspondentes dentro da base de dados. Para cada campo de cada uma das tabelas, é necessário determinar o tipo de dados que contem, para poder ajustar a estrutura da base de dados, e conseguir um armazenamento com a menor utilização de espaço. Este artigo descreve cada um dos tipos de dados que se podem ter num campo MySQL, a partir da versão 4.xx.xx.

Os tipos de dados que pode ter um campo, podem-se agrupar em três grandes grupos:

1. Tipos numéricos
2. Tipos de Data
3. Tipos de Cadeia

1 Tipos numéricos:

Existem tipos de dados numéricos, que se podem dividir em dois grandes grupos, os que estão em vírgula flutuante (com decimais) e os que não.

TinyInt: é um número inteiro com ou sem signo (sinal). Com signo a margem de valores válidos é desde -128 até 127. Sem signo, a margem de valores é de 0 até 255

Bit: um número inteiro que pode ser 0 ou 1.

SmallInt: número inteiro com ou sem signo (sinal). Com signo a margem de valores válidos é desde -32768 até 32767. Sem signo, a margem de valores é de 0 até 65535.

MediumInt: número inteiro com ou sem signo. Com signo a margem de valores válidos é desde -8.388.608 até 8.388.607. Sem signo, a margem de valores é de 0 até 16777215.

Int: número inteiro com ou sem signo. Com signo a margem de valores válidos é desde -2147483648 até 2147483647. Sem signo, a margem de valores é de 0 até 429.496.295

BigInt: número inteiro com ou sem signo. Com signo a margem de valores válidos é desde -9.223.372.036.854.775.808 até 9.223.372.036.854.775.807. Sem signo, a margem de valores é de 0 até 18.446.744.073.709.551.615.

Float: número pequeno em vírgula flutuante de precisão simples. Os valores válidos vão desde -3.402823466E+38 até -1.175494351E-38, 0 e desde 1.175494351E-38 até 3.402823466E+38.

Double: número em vírgula flutuante de dupla precisão. Os valores permitidos vão desde -1.7976931348623157E+308 até -2.2250738585072014E-308, 0 e desde 2.2250738585072014E-308 até 1.7976931348623157E+308

Decimal, Dec, Numeric: Número em vírgula flutuante desempacotado. O número armazena-se como uma cadeia.

Tipo de Campo	Tamanho de Armazenamento
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 ou 8 bytes
FLOAT	4 bytes

DOUBLE	8 bytes
DECIMAL(M,D)	M+2 bytes se $D > 0$, M+1 bytes se $D = 0$

2 Tipos data:

Na hora de armazenar datas, há que ter em conta que MySQL não verifica de uma maneira estrita se uma data é válida ou não. Simplesmente comprova que o mês está compreendido entre 0 e 12 e que o dia está compreendido entre 0 e 31.

Date: tipo data, armazena uma data. A margem de valores vai desde o 1 de Janeiro de 1001 ao 31 de dezembro de 9999. O formato de armazenamento é de ano-mes-dia.

DateTime: Combinação de data e hora. A margem de valores vai desde o 1 ed Janeiro de 1001 às 0 horas, 0 minutos e 0 segundos ao 31 de Dezembro de 9999 às 23 horas, 59 minutos e 59 segundos. O formato de armazenamento é de ano-mes-dia horas:minutos:segundos

TimeStamp: Combinação de data e hora. A margem vai desde o 1 de Janeiro de 1970 ao ano 2037. O formato de armazenamento depende do tamanho do campo:

Tamanho	Formato
14	AnoMesDiaHoraMinutoSegundo aaaammddhhmmss
12	AnoMesDiaHoraMinutoSegundo aammddhhmmss
8	AnoMesDia aaaammdd
6	AnoMesDia aammdd
4	AnoMes aamm
2	Ano aa

Time: armazena uma hora. A margem de horas vai desde -838 horas, 59 minutos e 59 segundos. O formato de armazenamento é 'HH:MM:SS'.

Year: armazena um ano. A margem de valores permitidos vai desde o ano 1901 ao ano 2155. O campo pode ter tamanho dois ou tamanho 4 dependendo de se queremos armazenar o ano com dois ou quatro algarismos.

Tipo de Campo	Tamanho de Armazenamento
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

3 Tipos de cadeia:

Char(n): armazena uma cadeia de longitude fixa. A cadeia poderá conter desde 0 até 255 caracteres. *Ocupa os 255 caracteres na memória, seja qual for o tamanho da cadeia.*

VarChar(n): armazena uma cadeia de longitude variável. A cadeia poderá conter desde

0 até 255 caracteres. **Ocupa somente o necessário, liberando memória.**

TinyText e TinyBlob: Coluna com uma longitude máxima de 255 caracteres.

Blob e Text: um texto com um máximo de 65535 caracteres.

MediumBlob e MediumText: um texto com um máximo de 16.777.215 caracteres.

LongBlob e LongText: um texto com um máximo de caracteres 4.294.967.295.

Enum: campo que pode ter um único valor de uma lista que se especifica. O tipo Enum aceita até 65535 valores diferentes.

Set: um campo que pode conter nenhum, um ou vários valores de uma lista. A lista pode ter um máximo de 64 valores.

Tipo de campo	Tamanho de Armazenamento
CHAR(n)	n bytes
VARCHAR(n)	n +1 bytes
TINYBLOB, TINYTEXT	Longitude+1 bytes
BLOB, TEXT	Longitude +2 bytes
MEDIUMBLOB, MEDIUMTEXT	Longitude +3 bytes
LOBLOB, LONGTEXT	Longitude +4 bytes
ENUM('value1','value2',...)	1 ó dos bytes dependendo do número de valores
SET('value1','value2',...)	1, 2, 3, 4 ó 8 bytes, dependendo do número de valores

Diferença de armazenamento entre os tipos Char e VarChar

Valor	CHAR(4)	Armazenamento	VARCHAR(4)	Armazenamento
"	"	4 bytes	"	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Conhecendo as bases de dados default do MYSQL:

- **Information_Schema** (Informações gerais sobre bases de dados, tabelas, campos e seus tipos, rotinas, procedimentos e mais uma infinidade de informações sobre os seu banco de dados. Entenderemos melhor esta base mais adiante.)
- **Mysql** (É a principal base de dados do seu Banco. Onde ficam armazenados seus usuários, premissões e até mesmo tópicos de ajuda sobre os comandos da linguagem SQL)

- **Test** (É uma base de dados default para que o usuário comece a fazer suas simulações)

❖ 2 – Manipulando dados com comandos da DML

DML (Linguagem de MANIPULAÇÃO de Dados)

→ **INSERT:** Este comando serve para você inserir informações distribuídas pelas colunas de sua tabela.

→ **SELECT:** Este comando serve para você listar os atributos desejados de determinada tabela em uma consulta.

→ **FROM:** Basicamente este comando especifica de onde se está se listando os atributos, ou seja, de qual ou quais tabelas eles estão sendo referenciados.

→ **WHERE:** Essa cláusula especifica uma ou mais condições que devem ser atendidas para que a listagem mostre o resultado desejado.

EXEMPLOS:

Considere a seguinte tabela:

cliente		
coluna	tipo de dado	extra
id (PK)	int(11)	Auto_increment
nome	varchar(60)	
sexo	enum('M','F')	
data_nascimento	date	
limite_de_credito	decimal(9, 2)	
cidade	varchar(50)	
bairro	varchar(50)	

INSERT

→ INSERÇÃO SIMPLES:

INSIRA na tabela cliente, nos campos (nome, sexo, data_nascimento, limite_de_credito) os seguintes valores ('FULANO', 'M', '1980-10-30', 3500.00)

```
INSERT INTO cliente(nome, sexo, data_nascimento, limite_de_credito)
VALUES('FULANO', 'M', '1980-10-30', 3500.00).
```

Importante: Observe que omitimos o campo id. Isto faz sentido, uma vez que seus valores são gerados automaticamente.

→ INSERÇÃO MÚLTIPLA:

```
INSERT INTO cliente(nome, sexo, data_nascimento, limite_de_credito)
VALUES('FULANO', 'M', '1980-10-30', 3500.00)
, ('BELTRANO', 'M', '1985-11-20', 4500.00)
, ('MARIA', 'F', '1980-07-06', 2000.00);
```

→ INSERÇÃO RESUMIDA:

INSERT INTO cliente VALUES(1,'FULANO', 'M', '1980-10-30', 3500.00);

Importante: Observe que neste caso fomos obrigados a fornecer o id e a colocar as informações na mesma ordem das colunas da tabela, embora não as tenhamos citado.

→ **INSERÇÃO RESUMIDA E MÚLTIPLA:**

INSERT INTO cliente VALUES
(1,'FULANO', 'M', '1980-10-30', 3500.00)
, (2, 'BELTRANO', 'M', '1985-11-20', 4500.00)
, (3, 'MARIA', 'F', '1980-07-06', 2000.00);

SELECT

→ **CONSULTAS SIMPLES:**

Selecione **TODOS OS CAMPOS** da tabela clientes:

- **SELECT * FROM** cliente.

Selecione os campos id e nome da tabela cliente:

- **SELECT** id, nome **FROM** cliente.

→ **CONSULTAS COM CONDIÇÕES:**

Selecione o código e a descrição dos clientes da cidade 'NOVA FRIBURGO':

- **SELECT** código, nome **FROM** cliente **WHERE** cidade='NOVA FRIBURGO'

Selecione **TODOS OS CAMPOS** da tabela cliente de todas as cidades, exceto 'NOVA FRIBURGO':

- **SELECT * FROM** cliente **WHERE** cidade <> 'NOVA FRIBURGO'

Selecione o nome dos clientes do bairro 'CENTRO' e da cidade 'NOVA FRIBURGO':

- **SELECT** nome **FROM** cliente **WHERE** bairro='CENTRO' **AND** cidade='NOVA FRIBURGO'

Selecione o nome dos clientes do bairro 'CENTRO' e das cidades 'NOVA FRIBURGO' OU 'CORDEIRO':

- **SELECT** nome **FROM** cliente **WHERE** bairro='CENTRO' **AND** (cidade='NOVA FRIBURGO' **OR** cidade='CORDEIRO')

NOTE QUE estou listando o nome de todos os clientes do CENTRO de NOVA FRIBURGO E do CENTRO de CORDEIRO.

Agora vamos supor que eu queira todos os clientes do bairro 'CENTRO' de **TODAS AS CIDADES, EXCETO 'CORDEIRO' E 'NOVA FRIBURGO'**. O comando ficaria assim:

- **SELECT** nome **FROM** cliente **WHERE** bairro='CENTRO' **AND NOT** (cidade='NOVA FRIBURGO' **OR** cidade='CORDEIRO')

Selecione todos os produtos **CUJO** custo é **MAIOR QUE 10**

- **SELECT * FROM** produto **WHERE** custo>10

Selecione todos os produtos CUJO custo é MAIOR OU IGUAL a 10

- **SELECT * FROM produto WHERE custo >= 10**

Selecione todos os produtos CUJO custo é MENOR QUE 20

- **SELECT * FROM produto WHERE custo < 20**

Nos exemplos acima, além da cláusula WHERE, usamos os OPERADORES BOLEANOS e os OPERADORES DE COMPARAÇÃO. São eles:

→ **OPERADORES BOLEANOS:**

- **AND (E)**
- **OR (OU)**
- **NOT (NEGAÇÃO) : Usado para negar uma ou mais condições.**

→ **OPERADORES DE COMPARAÇÃO:**

- **= IGUAL**
- **<> DIFERENTE**
- **> MAIOR QUE**
- **< MENOR QUE**
- **>= MAIOR OU IGUAL A**
- **<= MENOR OU IGUAL A**

→ **ORDENANDO CONSULTAS :**

Selecione todos os produtos CUJO fornecedor_id é IGUAL a 1 e o custo é MENOR OU IGUAL a 10, ordenado pelo custo (ordem decrescente):

```
SELECT * FROM produto  
WHERE fornecedor_id=1 AND custo <= 10  
ORDER BY custo DESC
```

Observe que um comando SQL tem uma ordem lógica:

Primeiramente você define sua seleção (SELECT)

Em seguida, define uma cláusula de condições usando os operadores (WHERE)

Por fim, ordena sua consulta (ORDER BY) e define se a ordem é ascendente (ASC) ou descendente (DESC).

IMPORTANTE: Se a ordem desejada for ascendente você **não precisa especificar** ASC. A ordem já é ascendente por default.

Você também pode ordenar por vários critérios. Exemplo:

```
SELECT * FROM produtos  
ORDER BY custo DESC, descricao
```

→ **Limitando a quantidade de registros retornados:**

Selecione todos os produtos CUJO fornecedor_id é IGUAL a 9 e o custo é MENOR OU IGUAL a 10, exibir apenas os 5 primeiros registros

SELECT * FROM produto WHERE fornecedor_id=9 AND custo<=10 ORDER BY descricao limit 5

Selecione todos os produtos CUJO fornecedor_id é IGUAL a 9 e o custo é MENOR OU IGUAL a 10, exibir apenas 5 registros, eliminando os 10 primeiros

SELECT * FROM produto WHERE codfornecedor=9 AND pcusto<=10 ORDER BY descricao limit 10,5

→ **Aplicando operações matemáticas para obter campos derivados**

Selecione descrição, preço de custo e preço de venda (campo virtual) de todos os produtos, considerando que a margem de lucro é 10%.

SELECT descrição, pcusto, pcusto*1.1 as pvenda FROM produto

→ **Comando DELETE:**

O uso do DELETE (excluir registro(s)) é bem simples e as condições podem ser as mesmas usadas num SELECT comum. Exemplo:

Apagar todos os clientes:

DELETE FROM cliente

Apagar todos os clientes da cidade 'BOM JARDIM':

DELETE FROM cliente WHERE cidade='BOM JARDIM'

→ **Comando UPDATE:**

Atualizar em 10% o custo de todos os produtos:

UPDATE produto SET custo=custo*1.1

Atualizar, diminuindo em 10%, o custo e venda de todos os produtos com estoque acima de 100:

UPDATE produto set custo=custo*0.9, venda=venda*0.9
where quantidade>100

Atualizar o cadastro de clientes mudando o nome da cidade de 'FRIBURGO' para 'NOVA FRIBURGO':

UPDATE cliente set cidade='NOVA FRIBURGO' WHERE cidade='FRIBURGO'