

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/341327421>

Introduction to Database

Book · January 2015

CITATIONS

0

READS

147

1 author:



[Alvaro F. Pinheiro](#)

University of Pernambuco

29 PUBLICATIONS 5 CITATIONS

SEE PROFILE

Fundamentos da Engenharia de Software

Introdução à Banco de Dados

PINHEIRO, Álvaro Farias
Autor

Fundamentos da Engenharia de Software

Introdução à Banco de Dados

Publicação 2015

O autor acredita que todas as informações aqui apresentadas estão corretas e podem ser utilizadas para qualquer fim legal. Entretanto, não existe qualquer garantia explícita ou implícita, de que o uso de tais informações conduzirá sempre ao resultado desejado. Os nomes de sites e empresas, por ventura, mencionados, foram utilizados apenas para ilustrar os exemplos, não tendo vínculo nenhum com o livro, não garantindo a sua existência nem divulgação. Eventuais erratas estarão disponíveis para download no site de publicação.

Algumas imagens utilizadas neste livro foram obtidas a partir da Internet.

Dados da Publicação

Pinheiro, Álvaro Farias

Fundamentos da Engenharia de Software: Introdução à Banco de Dados

1. ed. – Volume II

Recife, 2015

Selo Editorial: Publicação Independente

Bibliografia

ASIN B0138W79R8

ISBN 9781520120454

1. Banco de Dados
2. Modelagem
3. Normalização
4. SGBD
5. SQL

Fundamentos da Engenharia de Software

Introdução à Banco de Dados

Publicação Independente
Livro eletrônico em português com o título
Fundamentos da Engenharia de Software
Introdução à Banco de Dados
Volume 2
1ª Edição
Recife - Pernambuco – Brasil
Primeira revisão em Outubro de 2015
Segunda revisão em Junho de 2016

Fundamentos da Engenharia de Software

Introdução à Banco de Dados

Aos meus irmãos Alexandre Farias da Luz e Maria Alice Farias de Moraes por manterem os laços de família.

Fundamentos da Engenharia de Software

Introdução à Banco de Dados

“Sem dados não se obtém informação.
Sem informação não se tem conhecimento”
Christopher J. Date

SUMÁRIO

1.	Apresentação	2
1.1	Apresentação do Modelo Hierárquico	5
1.2	Apresentação do Modelo Rede	6
1.3	Apresentação do Modelo Relacional	8
1.4	Apresentação do Modelo Objeto Relacional	8
2.	Introdução ao Modelo Relacional	10
2.1	Structure Query Language (SQL)	10
2.1.1	Inserção de Metadados (Criação de Tabelas)	11
2.1.2	Inserção de Dados (Linhas Registros Ocorrências Tuplas)	13
2.1.3	Alteração de Metadados (Modificar Estruturas)	15
2.1.4	Alteração de Dados (Atualização)	17
2.1.5	Algumas Funções	20
2.1.6	Índices	21
2.1.7	Dicionário de Dados	22
2.1.8	Consulta de Dados	22
2.1.9	Precedência	24
2.1.10	Comparação	24
2.1.11	Projeção	24
2.1.12	Seleção	24
2.1.13	Junção	24
2.1.14	Consulta Básica	25
2.1.15	Apelido (Alias)	25
2.1.16	Campos Calculados	26
2.1.17	Valores Nulos	26
2.1.18	Duplicidade	27
2.1.19	Condicionamento	28
2.1.20	Ordenação	30
2.1.21	Aninhamento	30
2.1.22	Funções	31
2.1.23	Funções Caractere	32
2.1.24	Funções Numérica	33

SUMÁRIO

2.1.25 MOD %	34
2.1.26 Funções Data	34
2.1.27 Função Conversão	36
2.1.28 Função de Busca e Troca	36
2.1.29 Função de Condicionamento	37
2.1.30 Produto Cartesiano	37
2.1.31 Junção	37
2.1.32 Equijunção	38
2.1.33 Não-equijunção	39
2.1.34 Junção Externa	39
2.1.35 Autojunção	39
2.1.36 Funções de Grupo	40
2.1.37 Grupos de Dados	40
2.1.38 Grupo de Dados com Filtro	41
2.1.39 ROLLUP	42
2.1.40 CUBE	43
2.1.41 Ordenação de Dados	43
2.1.42 Aninhamento de Grupo de Dados	44
2.1.43 Subconsulta	44
2.1.44 ANY	45
2.1.45 ALL	45
2.1.46 Correlacionadas	46
2.1.47 EXISTS	46
2.1.48 União	47
2.1.49 Intersecção	47
2.1.50 Subtração	47
2.1.51 Macrossubstituição	48
2.1.52 Seleção de Seleção	48
2.1.53 Análise Top-N	49
2.1.54 Excluir Registros	49
2.1.55 Excluir e recriar tabela:	49
2.1.56 Excluir Tabela	49
2.1.57 Transações	50

SUMÁRIO

2.1.58 Visão	50
2.1.59 Controle de Usuários.....	52
2.1.60 Privilégios	52
2.1.61 Sequências.....	53
2.1.62 Vínculos	53
2.2 Objetos de Bancos de Dados	53
2.3 Principais Funções do SQLServer.....	54
2.4 Tipos de Dados	61
2.5 Usuários e Senhas Padrões.....	63
2.6 Exemplos.....	63
2.7 Exercícios.....	64
3.1 Apresentações.....	79
3.2 Aulas.....	79
3.3 Ferramentas	79
3.4 Sugestões de Leitura.....	79
4.1 Referências Web	80
4.2 Referências Bibliográficas	80

ILUSTRAÇÕES

Figura 1.1 Dados	2
Figura 1.2 Metadado.....	2
Figura 1.3 Informação.....	3
Figura 1.4 Conhecimento.....	3
Figura 1.5 Dimensões dos Dados.....	5
Figura 1.6 Modelo Hierárquico.....	6
Figura 1.7 Modelo Rede	7
Figura 1.8 Modelo Relacional	8
Figura 2.1 Modelo Entidade Relacionamento	73

SIGLAS e ACRÔNIMOS

ANSI - *American National Standards Institute*

API - *Application Programming Interface*

BI - *Business Intelligence*

DBMS - *DataBase Management System*

DBTG- *Data Base Task Group*

IMS - *Information Management System*

ISO - *International Organization for Standardization*

OLAP - *OnLine Analytical Processing*

OLPT - *OnLine Transaction Processing*

SQL - *Structured Query Language*

NOTAÇÕES

Este livre adota as seguintes notações, que devem ser observadas para o perfeito entendimento do material aqui produzido.

Notações Técnicas:

<..> - Menor que / Maior que, nos comandos SQL usado para metadados.

<..> - Menor / Maior, no texto indica que é um *hyperlink* (*link* para um endereço web).

[..] - Abre e fecha parênteses, no texto indica utilização de uma sigla.

(..) - Abre e fecha colchetes, nos comandos SQL indica opcionalidade.

| - Barra, para indicar “ou lógico”.

 - Sublinhado, nos comandos SQL indica o *default* (padrão).

 - Sublinhado, no texto indica que é um *hyperlink* (*link* para um endereço web).

Negritos - Títulos de capítulos, seções e subseções.

Itálico - Para palavras de língua estrangeira.

SOBRE O AUTOR



Álvaro Farias Pinheiro tem mais de 30 anos de experiência no campo da computação e nestes, mais de 15 anos na área acadêmica, lecionando disciplinas com foco em engenharia de software.

É mestre em Engenharia de Software pelo Centro de Estudos e Sistemas Avançados do Recife (CESAR) com a dissertação *Uso do Balanced Scorecard na Avaliação de Fábricas de Software* utilizadas como Instrumento de Ensino de Tecnologia da Informação Baseado em Problemas Reais. Especialista em Gestão Pública pela

Universidade de Pernambuco (UPE) e em Metodologias de Desenvolvimento pela União Brasileira de Tecnologia. É Bacharel em Sistemas de Informação pela Faculdade Integrada do Recife (FIR), iniciando sua vida acadêmica no Bacharelado em Ciência da Computação da Universidade Católica de Pernambuco (UNICAP).

Iniciou as atividades profissionais em 1986 como Programador na Elógica Microsistemas desenvolvendo para baixas e altas plataformas nas linguagens dBase, Clipper, FoxPro, Cobol e Natural. Como Analista de Sistemas passou pelas instituições: Centro de Prestação de Serviços Técnicos de Pernambuco (CETEPE), Empresa de Fomento da Informática do Estado de Pernambuco (FISEPE), Instituto de Tecnologia em Informática (ITECI), Apply Solutions e está hoje na Agência Estadual de Tecnologia da Informação do Estado de Pernambuco (ATI-PE) por concurso público, trabalhando como Analista em Gestão da Tecnologia da Informação e Comunicação, em exercício na Procuradoria Geral do Estado de Pernambuco (PGE-PE), atuando como Coordenador da Área de Sistemas.

Começou a lecionar em 1992 em cursos técnicos nas empresas ITECI e IBRATEC, e desde 2006 é professor de cursos de graduação e pós-graduação em computação nas instituições de ensino superior da região, a citar, Universidade Federal Rural de Pernambuco (UFRPE), Instituto Federal de Pernambuco (IFPE), Faculdade Estácio do Recife, Faculdade Integrada do Recife (FIR), Faculdade Joaquim Nabuco, Faculdade do Recife, Unibrattec e durante esses anos realizou diversos cursos de extensão como os de Análise de Ponto de Função, Administração de Redes, Linguagens de Programação Natural, Centura, Delphi e CSharp, Banco de Dados Adabas, MySQL, Postgre e Oracle, Análise Estruturada e Orientada a Objetos, entre outros.

Possui experiência na área de Engenharia de *Software* nas disciplinas de Teoria da Computação, Sistemas de Informação e Colaborativos, Desenvolvimento Estrutural e Orientado a Objetos para aplicações Desktop, Web e Mobile, Processos de Desenvolvimento IRUP, XP e Scrum, Banco de Dados SQLServer e Oracle, e Gestão de Projetos com PMBOK e SWEBOK. Trabalha atualmente

SOBRE O AUTOR

na coordenação de projetos e com educação presencial e a distância, ministrando aulas e realizando pesquisas em ambientes virtuais de ensino e linguagens de programação.

<http://www.alvarofpinheiro.eti.br>

<http://br.linkedin.com/in/alvarofpinheiro>

<https://sites.google.com/site/alvarofpinheiro>

<http://www.facebook.com/alvarofpinheiroaulas>

PREFÁCIO

O motivo que me levou a escrever este livro foram os meus alunos que ao longo dos anos ministrando aulas e sempre escrevendo nos quadros o conteúdo passado e também disponibilizando para os discentes materiais complementares para auxiliar o acompanhamento do assunto ministrado, sempre era abordado por um ou outro me indagando porque não disponibilizado esse material num livro. A resposta sempre era a mesma. Falta de tempo, mas não de interesse.

Ao passar dos anos o volume de material produzido para as diversas aulas ministradas e que ainda ministro sobre engenharia de *software* contemplando os temas que são citados nos capítulos foram se avolumando, chegando ao ponto que após uma compilação daria para produzir um livro. Assim depois de muitos pedidos estou disponibilizando esse material compilado e integrado.

Esse livro é composto de aulas que foram produzidas de forma independente e que foram ministradas em momentos distintos. Porém todas sobre um tema comum, a engenharia de *software*, pois essa foi minha formação profissional e acadêmica. Tendo iniciado a vida profissional como programador e passando depois de todos os degraus, junior, pleno e sênior, para analista de sistemas, e mais uma vez depois dos degraus para coordenador de projetos, sempre em paralelo busquei o aperfeiçoando dos conhecimentos adquiridos nos trabalhos com os obtidos no mundo acadêmico, através de cursos técnicos, graduações, pós-graduações e extensões.

Deste modo, comecei a lecionar aplicando nas aulas todo esse *know-how* adquirido. Como uma pessoa que sempre gostou de ler e escrever, antes de lecionar uma nova disciplina a qual tinha sido convidado, sempre preparei antes as aulas, construindo além do seu plano de aula baseado em suas ementas, também o conteúdo que seria ministrado aula a aula.

Este material foi montado através da compilação das diversas aulas que ministrei durante esses anos de docência. Quanto da integração desses conteúdos, o fiz, seguindo a lógica de obtenção dos conhecimentos necessários para fundamentar o seguinte, assim, sugiro que a leitura deste livro seja feita de forma sequencial para melhor aproveitamento dos assuntos.

Finalizando, este livro possui várias subseções que disponibilizam *hyperlink's* para *wiki's* sobre os assuntos abordados, como também *link's* para outros *ebook's* utilizados como referencial bibliográfico para o conteúdo produzido. Também *Links* para os sites oficiais dos temas estudados, alguns com *Links* para vídeos, além de atividades propostas, exemplos e biografia dos principais estudiosos da tecnologia da informação e comunicação.

PREFÁCIO

Álvaro Farias Pinheiro

Introdução à Banco de Dados. Neste são abordados os conceitos básicos de bancos de dados e seus sistemas gerenciadores, mas com o foco na arquitetura relacional, porque ainda hoje o mercado faz uso em larga escala desses bancos de dados, mesmo que o paradigma predominante seja o orientado a objetos e que, já existam a um bom tempo bancos orientados a objeto, até mesmo os bancos objetos-relacionais que são um híbrido entre essas duas arquiteturas, o que predomina ainda é o relacional, assim, este material é focado na linguagem de consulta estruturada para os SGBD-Rs do mercado, com foco na comparação de cinco dos mais utilizados bancos relacionais, os quais são: Oracle, SQLServer, MySQL, SQLBase e Interbase.

1.Apresentação

A importância de armazenar, organizar e acessar dados não é de agora, já de muito tempo existe essa necessidade, pois guardar de forma segura dados úteis que no futuro possam transforma-se em informação é uma premissa muito antiga da humanidade. Mas, nos dias atuais isso se tornou algo fundamental e tê-lo de forma bem estruturada pode ser um diferencial entre o sucesso ou não de um negócio.

Observe que no parágrafo anterior foi especificado dois termos que muitos usam como sinônimos, mas academicamente possuem significados diferentes. Dado e Informação. Para iniciarmos o estudo de Banco de Dados vamos entender os conceitos de Dados, Metadado, Informação e Conhecimento. Observe a figuras 1.1, 1.2, 1.3 e 1.4 para entender a diferença desses termos.

Dado é um valor armazenado que por si só não nos fornece um significado. Observe a figura 1.1 os valores 10, 11, 13, 20 e 21 não nos fornecem nenhum significado.

Figura 1.1 Dados

10
11
13
20
21
25
30
47
48

Fonte: Próprio Autor

Metadado é o dado que designa outro dado, isto é, o identifica, explicando através de um nome o seu significado. Olhe a figura 1.2 para entender melhor o que é um metado. Os valores 10, 11, 13, 20 e 21 são os dados da coluna ou campo denominado IDADE, assim sendo IDADE é um metadado que identifica os dados armazenados no campo.

Figura 1.2 Metadado

IDADE
10
11
13

20
21
25
30
47
48

Fonte: Próprio Autor

Informação é o dado processado, isto é, é o resultado do processamento dos dados 10, 11, 13, 20 e 21 respondendo a uma pergunta. Por exemplo, quais as idades acima de 18 anos armazenadas? Resposta decorrente do processamento dos dados 10, 11, 13, 20, 21 são 2 idades, isto é, as de 20 e 21 anos. Que correspondem as idades acima de 18 anos.

Figura 1.3 Informação

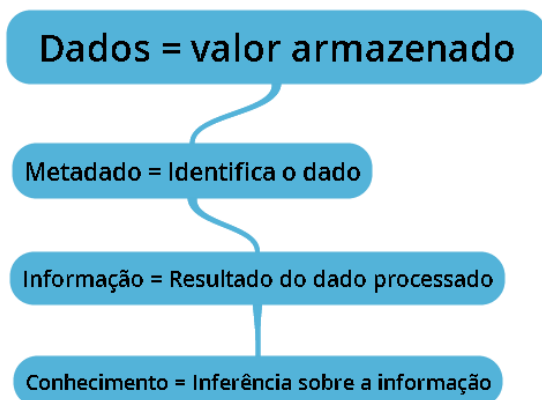
IDADE
10
11
13
20
21
25
30
47
48
6 idades acima de 18 anos

Fonte: Próprio Autor

Conhecimento é o que se deseja com uma informação. Neste exemplo sabemos que apenas 2 idades das 5 armazenadas estão acima de 18 anos. Podemos dizer que se o objetivo é saber quantas pessoas cadastradas são de maiores, esse objetivo foi atingido, assim podemos dizer que temos conhecimento desse negócio.

Resumindo. Para sabermos algo precisamos de informação e para se obter a informação necessitamos de Dados, pois os Dados processados geram Informação e com a Informação obtem-se o Conhecimento.

Figura 1.4 Conhecimento



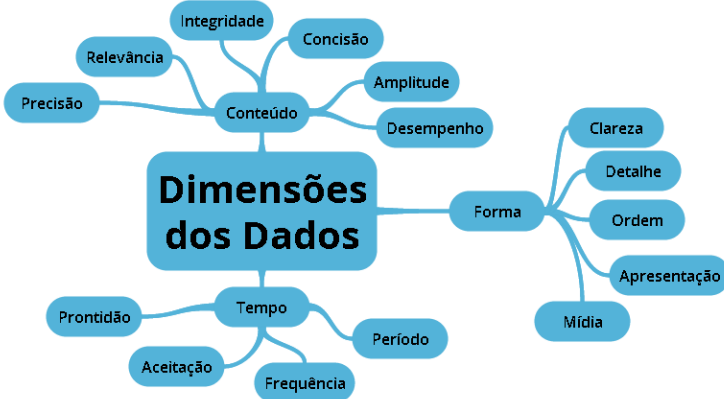
Fonte: Próprio Autor

Sobre dados ainda temos segundo O'Brien (2004) que os dados possuem dimensões três dimensões, veja a figura 1.5. A dimensão do Tempo, a do Conteúdo e a da Forma. A dimensão do tempo se refere: Prontidão que se refere a informação que deve ser fornecida quando necessária; Aceitação que deve ser atualizada quando fornecida; Frequência que deve ser fornecida quantas vezes for necessária; e Período que deve se referir ao passado, presente e futuro. A dimensão do Conteúdo fala sobre: Precisão que deve estar isenta de erros; Relevância que deve estar relacionada as necessidades de informação de um receptor específico para uma situação específica; Integridade que toda informação necessária deve ser fornecida; Concisão que apenas a informação necessária deve ser fornecida; Amplitude a informação pode ter um alcance amplo ou estreito, ser interna ou externa; e Desempenho deve ser sobre atividades concluídas, processos realizados ou recursos acumulados. Sobre a dimensão da Forma se comunica sobre: Clareza que a informação deve ser fornecida sobre uma forma que seja fácil de compreender; Detalhe que pode ser fornecida de forma detalhada ou resumida; Ordem deve ser organizada; Apresentação deve ser apresentada em forma de narrativa, numérica, gráfica ou outras; e Mídia que deve ser fornecida em alguma mídia de conhecimento geral.

Para continuarmos o entendimento sobre banco de dados vamos agora falar sobre sua evolução. Mas antes vamos entender o que é um Sistema Gerenciador de Banco de Dados (SGBD) que no inglês é conhecido com Data Base Management System (DBMS). Eles servem para gerenciar uma base de dados. Um SGBD trata-se de um conjunto de programas (*Application Programming Interface*) mais conhecidos pela sigla API que objetivam retirar de um aplicação desenvolvida em alguma linguagem de programação a responsabilidade de gerenciar o acesso a um banco de dados qualquer, como também a manutenção e obtenção dos dados. Assim os SGBDs disponibilizam uma interface, mais conhecidos como drivers de conexão com banco de dados,

que são utilizadas pelos usuários (clientes) para realizar operações como inclusão, alteração, exclusão ou consulta de dados.

Figura 1.5 Dimensões dos Dados



Fonte: Próprio Autor

Um dos primeiros SGBDs utilizados comercialmente apareceu no mercado no final da década de sessenta e era baseado no primeiros tipos de arquivos de armazenamento chamdos de Sistemas de Arquivos. Esses de Sistemas de Arquivos não controlavam acesso concorrente, isto é, acesso de vários usuários ou processos ao mesmo tempo. A evolução dos bancos de dados aconteceu a partir desses Sistemas de Arquivos tornando-se os SGBD's que atualmente são utilizados. Mas essa evolução possou por diferentes formas de representação dos dados, melhor dizendo, diferentes estruturas de armazenamento de dados.

A evolução dos SGBD's em relação aos modelos de dados em ordem sequencial se deu da seguinte forma: Modelo de Arquivos, Modelo Hierárquico, Modelo em Redes, Modelo Relacional, Modelo Orientado a Objetos e Modelo Objeto-Relacional.

1.1 Apresentação do Modelo Hierárquico

Depois do Sistemas de Arquivos que não possuía uma SGBD o primeiro modelo que foi desenvolvido com um SGBD foi o modelo hierárquico. Podemos dizer que seu desenvolvimento só foi possível graças a tecnologia dos discos de armazenamento endereçáveis, isto devido ao fato que esses disco possuíam a estrutura de endereçamento físico que viabilizavam a representação hierárquica das informações.

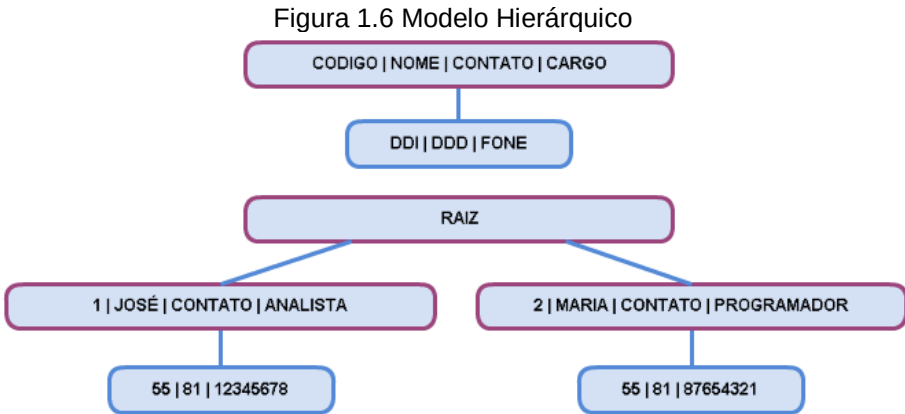
Nesse modelo os dados são organizados em árvores, onde os nós árvores contêm as ocorrências dos registros e estes por sua vez são uma coleção de

campos, onde o registro da hierarquia anterior é o registro-pai e o demais são os registros-filhos.

O relacionamento entre um registro-pai e vários registros-filhos é caracterizado pelo que é chamado de cardinalidade 1:N (lê-se 1 para N) onde um registro (ocorrência) pode estar relacionado a vários outros registros (ocorrências).

Um dos principais bancos desse modelo de banco de dados foi o *Information Management System (IMS)* da *IBM Corporation*. Mas essa arquitetura de banco possui algumas grandes desvantagens como a replicação dos dados que pode causar inconsistência de dados na atualização, outra é o desperdício de espaço em disco, e talvez a mais agravante é que grande parte das restrições dos dados como também as consistências eram escritas nas aplicações, sendo assim necessário codificar programas muito extensos para acessar, controlar e manipular o banco de dados.

A figura 1.6 mostra um exemplo de esquema para um banco de dados hierárquico baseado em dois componentes os quais são: caixas que correspondem aos registros ou ocorrências; e Ligações (linhas) que servem para representar às relações entre os tipos de registros.



Fonte: Próprio Autor

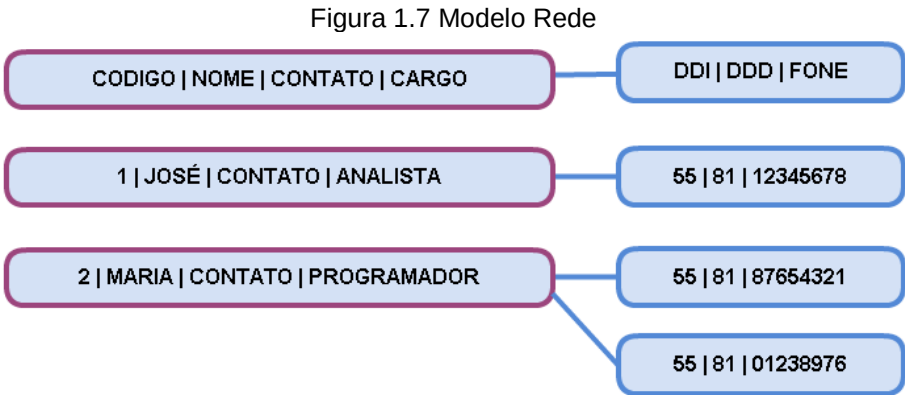
1.2 Apresentação do Modelo Rede

A evolução do Modelo Hierárquico foi o Modelo Rede podemos dizer que esse último é uma extensão do primeiro, pois o Modelo Rede eliminou o conceito de hierarquia permitindo que um registro tivesse várias associações. Nesse modelo, os registros são organizados em grafos existindo um único tipo de associação, chamado de conjunto ou set definindo a relação 1:N (1 para N) entre esses dois tipos de registros chamados de proprietário e membro.

Um exemplo de banco de dados que foi construído com base no modelo de redes foi o *Data Base Task Group* (DBTG). Este banco permite o relacionamento de N registros de uma entidade com N outros registros de outra entidade permitindo assim o relacionamento M:N (M para N ou muitos para muitos).

O uso desse modelo permitiu a garantia para seus usuários de dois aspectos importantes o de concorrência e segurança. Em relação a segurança o banco permitia que parte do banco de dados (área) pudesse ser bloqueada permitindo acessos simultâneos, concorrência. Ao contrário do Modelo Hierárquico que permitia acesso aos dados apenas pela raiz, o Modelo em Rede permitia acesso a qualquer nó da rede sem ter que passar pela raiz.

A figura 1.7 mostra a representação do Modelo em Redes o qual consiste em dois componentes básicos: Registros e Linhas.



Fonte: Próprio Autor

Os modelos hierárquico e rede são modelos denominados Orientados a Registros, isto é, qualquer acesso à base de dados para realizar as manutenções, a exemplo, inclusão, alteração, exclusão e consulta, só pode ser realizado uma operação por vez, melhor dizendo, um registro de cada vez.

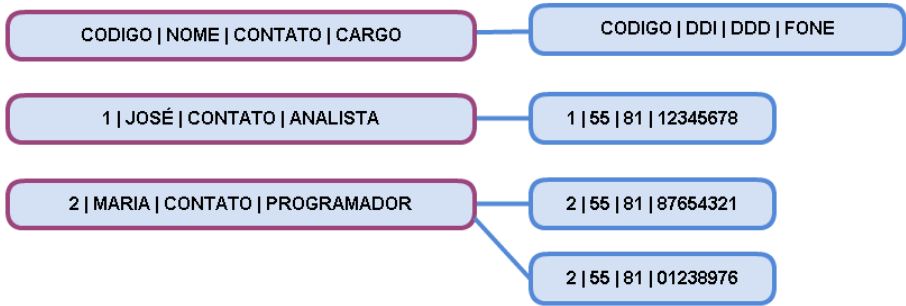
1.3 Apresentação do Modelo Relacional

Com o passar do tempo o Modelo em Rede naturalmente sofreria necessidade de evolução e como o aumento do desejo por parte dos administradores de base de dados em obter maior independência de dados, surgiu um conjunto de funções baseadas na álgebra relacional com o intuito de armazenar e recuperar dados.

O modelo relacional, tem como base a teoria dos conjuntos e álgebra relacional e foi resultado de um estudo teórico da lógica de predicados e na teoria dos conjuntos realizado por Edgar Frank Codd em 1970, quando o mesmo publicou o artigo "*Relational Model of Data for Large Shared Data Banks*". Sendo o primeiro modelo de dados descrito formalmente fez com que os outros , passassem a serem conhecidos como modelos hierárquicos, modelos em rede.

A figura 1.8 mostra um exemplo de esquema para um banco de dados relacional que é basicamente composto de dois componentes: a Entidade e a Relação, onde a entidade é um elemento caracterizado pelos dados que são elicitados e documentados na especificação de requisitos e são comumente chamados de tabela; a relação determina o modo como cada registro de uma tabela se relaciona com registros de outras tabelas.

Figura 1.8 Modelo Relacional



Fonte: Próprio Autor

O modelo relacional é o sucessor natural do modelo hierárquico e do modelo em rede, porém todas essas arquiteturas são usadas até os dias de hoje pois para muitos sistemas antigos com grande quantidade de dados a migração não é algo viável devido ao alto custo que demandaria.

1.4 Apresentação do Modelo Objeto Relacional

Com a aceitação pela comunidade tecnológica do paradigma orientado a objetos tanto para programação como para a análise inevitavelmente surgiram

novos modelos baseados em orientação ao objeto desenvolvidos, a exemplo de banco de dados. Porém a explicação deste será em outra abordagem já que o foco deste livro é o modelo relacional detalhado a seguir.

2.Introdução ao Modelo Relacional

O modelo relacional foi desenvolvido por Frank Codd que fez uso da teoria matemática dos conjuntos para representar as operações de manipulação de dados de forma mais simples e rápida.

Esse modelo de banco de dados utiliza a linguagem denominada de Linguagem de Consulta Estruturada que é mais conhecida pela sua sigla em inglês SQL acrônimo de *Structured Query Language*. Mesmo com a existência de um novo modelo com base no paradigma orientado a objetos, ela é ainda predominante, pois continua sendo muito popular entre as linguagens de banco de dados.

O objetivo dessa disciplina é introduzir nos conceitos de bancos de dados em especial banco de dados relacionais, focando na Linguagem de Consulta Estruturada que é uma linguagem do paradigma declarativo para banco de dados relacional baseadas na álgebra relacional.

O SQL surgiu na década de 70 na IBM baseado no modelo relacional de Codd. É o padrão para banco de dados relacionais e é mantido pela *American National Standards Institute* (ANSI) e *International Standards Organization* (ISO). Existindo várias versões revisadas do SQL ANSI/ISO, as quais são: 1986, 1987, 1992, 1999 e 2003. E além dessas padronizações, existem as extensões inseridas pelos desenvolvedores de banco de dados.

Pode ser utilizado por uma grande faixa de usuários, mesmo com pouca experiência em programação e é uma linguagem não procedural que reduz o tempo necessário para a criação e manutenção de sistemas e é uma linguagem similar ao inglês, facilitando o seu aprendizado.

Como essa disciplina é extremamente prática a forma de transmissão do assunto será com muitos exemplos, isto é, mostra-se a teoria com base na sintaxe e em seguida um ou mais exemplos.

2.1 Structure Query Language (SQL)

Iniciando o estudo do SQL temos as seguintes categorias:

- *Data Definition Language* (DDL): Create ..., Drop ..., Alter ...;
- *Data Manipulation Language* (DML): Insert ..., Update ..., Delete ...;
- *Data Query Language* (DQL): Select ...;
- *Data Transaction Language* (DTL): Savepoint ..., Rollback ..., Commit ...;
- *Data Control Language* (DCL): Grant ..., Revoke ...;

2.1.1 Inserção de Metadados (Criação de Tabelas)

Comando que permite criar as estruturas de armazenamento de dados.

2.1.1.1 Sintaxe

```
CREATE TABLE nome_tabela [(definição_create,...)] [table_options]
[select_statement]
```

definição_create:

```
nome_coluna tipo [NOT NULL | NULL] [DEFAULT valor_padrão]
[[PRIMARY] KEY]
| [CONSTRAINT [symbol]] PRIMARY KEY (index_col_name,...)
| INDEX [nome_indice] (index_nome_coluna,...)
| [CONSTRAINT [symbol]] UNIQUE [INDEX] [index_name] (index_col_name,...)
| [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (index_col_name,...)
```

definição_referência:

```
REFERENCES nome_tabela [(index_nome_coluna,...)]
[ON DELETE opção_referência]
[ON UPDATE opção_referência]
```

opção_referência:

```
RESTRICT | CASCADE | SET NULL | SET DEFAULT
```

2.1.1.2 Dicas

É possível criar uma tabela a partir de uma subconsulta;

```
CREATE TABLE tabela [coluna, ...] AS subconsulta.
```

2.1.1.3 Práticas

Criar as seguintes tabelas: Tmp: Número (PK), número inteiro. Cargo: Número (PK), número inteiro, auto-incremento; Nome, literal variável(30), não nulo. Departamento: Número (PK), número inteiro; Nome, literal variável(30), não nulo; Fator, número decimal com precisão 3 e escala 2. Empregado: Número (PK), número inteiro; Nome, literal variável (50), não nulo; Departamento, (FK) numérico, atualização cascata, exclusão restrita.

--SQLServer

```
CREATE TABLE CARGO (
  NUMERO INTEGER IDENTITY( 1, 1),
  NOME VARCHAR(30) NOT NULL,
  PRIMARY KEY (NUMERO)
);
```

--Postgre|Oracle

```
CREATE TABLE CARGO (
  NUMERO INTEGER,
  NOME VARCHAR(30) NOT NULL,
  PRIMARY KEY (NUMERO)
);
```

```
CREATE SEQUENCE CARGOSEQ INCREMENT BY 1 START WITH 1;
```

```
--MySQL
```

```
CREATE TABLE CARGO (  
  NUMERO INTEGER AUTO_INCREMENT,  
  NOME VARCHAR(30) NOT NULL,  
  PRIMARY KEY (NUMERO)  
);
```

```
--Firebird
```

```
CREATE TABLE CARGO (  
  NUMERO INTEGER,  
  NOME VARCHAR(30) NOT NULL,  
  PRIMARY KEY (NUMERO)  
);  
CREATE GENERATOR CARGOGEN;  
CREATE TRIGGER CARGOTRI FOR CARGO  
ACTIVE BEFORE INSERT POSITION 0  
AS  
BEGIN  
  NEW.NUMERO = GEN_ID(CARGOGEN, 1);  
END;  
CREATE TABLE TMP (  
  NUMERO INTEGER PRIMARY KEY  
);
```

```
--Todos
```

```
CREATE TABLE DEPARTAMENTO (  
  NUMERO INTEGER,  
  NOME VARCHAR(30) NOT NULL,  
  FATOR DECIMAL(3,2),  
  PRIMARY KEY (NUMERO)  
);
```

```
--SQLServer | Postgre | MySQL | Firebird
```

```
CREATE TABLE EMPREGADO (  
  NUMERO INTEGER,  
  NOME VARCHAR(50) NOT NULL,  
  DEPARTAMENTO INTEGER,  
  PRIMARY KEY (NUMERO),  
  FOREIGN KEY (DEPARTAMENTO) REFERENCES DEPARTAMENTO  
  (NUMERO) ON UPDATE CASCADE  
);
```

```
--Oracle
```

```
CREATE TABLE EMPREGADO (  
  NUMERO INTEGER,  
  NOME VARCHAR(50) NOT NULL,  
  DEPARTAMENTO INTEGER,  
  PRIMARY KEY (NUMERO),  
  FOREIGN KEY (DEPARTAMENTO) REFERENCES DEPARTAMENTO  
  (NUMERO)  
);
```

2.1.2 Inserção de Dados (Linhas | Registros | Ocorrências | Tuplas)

Comando que permite incluir os dados nas estruturas de armazenamento.

2.1.2.1 Sintaxe

```
INSERT [INTO] nome_tabela [(nome_coluna,...)]  
  VALUES ((expressão | DEFAULT),...),(...),...  
  | [INTO] nome_tabela [(nome_coluna,...)]  
  SELECT ...
```

2.1.2.2 Dicas

Pode-se inserir linha a linha;

Pode-se dependendo do BD inserir uma lista;

Pode-se inserir linhas de uma tabela em outra;

Não se usa a cláusula VALUE e sim SELECT.

2.1.2.3 Práticas

Inserir esses dados nas tabela Cargo: Número, Nome; 1, Analista Trainee; 2, Analista Junior; 3, Analista Pleno; 4, Analista Sênior; 5, Analista Master:

--SqlServer | MySql | Firebird

```
INSERT INTO CARGO (NOME) VALUES ('ANALISTA TRAINEE');  
INSERT INTO CARGO (NOME) VALUES ('ANALISTA JUNIOR');  
INSERT INTO CARGO (NOME) VALUES ('ANALISTA PLENO');  
INSERT INTO CARGO (NOME) VALUES ('ANALISTA SÊNIOR');  
INSERT INTO CARGO (NOME) VALUES ('ANALISTA MASTER');  
INSERT INTO CARGO (NOME) VALUES ('CONSULTOR');
```

--Postgre

```
INSERT INTO CARGO (NUMERO, NOME) VALUES (nextval('CARGOSEQ'),  
'ANALISTA TRAINEE');  
INSERT INTO CARGO (NUMERO, NOME) VALUES (nextval('CARGOSEQ'),  
'ANALISTA JUNIOR');  
INSERT INTO CARGO (NUMERO, NOME) VALUES (nextval('CARGOSEQ'),  
'ANALISTA PLENO');  
INSERT INTO CARGO (NUMERO, NOME) VALUES (nextval('CARGOSEQ'),  
'ANALISTA SÊNIOR');
```

```
INSERT INTO CARGO (NUMERO, NOME) VALUES (nextval('CARGOSEQ'),  
'ANALISTA MASTER');  
INSERT INTO CARGO (NUMERO, NOME) VALUES (nextval('CARGOSEQ'),  
'CONSULTOR');
```

--Oracle

```
INSERT INTO CARGO (NUMERO, NOME) VALUES (CARGOSEQ.NextVal,  
'ANALISTA TRAINEE');  
INSERT INTO CARGO (NUMERO, NOME) VALUES (CARGOSEQ.NextVal,  
'ANALISTA JUNIOR');  
INSERT INTO CARGO (NUMERO, NOME) VALUES (CARGOSEQ.NextVal,  
'ANALISTA PLENO');  
INSERT INTO CARGO (NUMERO, NOME) VALUES (CARGOSEQ.NextVal,  
'ANALISTA SÊNIOR');  
INSERT INTO CARGO (NUMERO, NOME) VALUES (CARGOSEQ.NextVal,  
'ANALISTA MASTER');  
INSERT INTO CARGO (NUMERO, NOME) VALUES (CARGOSEQ.NextVal,  
'CONSULTOR');
```

Departamento: Número, Nome, Fator; 10, Administração (Admin), 1.75; 20, Marketing (Mkt), 1.85; 50, Compras (Com), 2.73; 60, Tecnologia (Tec), 1.55; 80, Vendas (Ven), 1.87; 90, Executivo (Exec), 1.92; 110, Contabilidade (Contab), 2.35; 190, Contratos (Ctt), 2.23;

--Todos

```
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (10,  
'ADMINISTRAÇÃO (ADMIN)', 1.75);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (20,  
'MARKETING (MKT)', 1.85);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (50,  
'COMPRAS (COM)', 2.73);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (60,  
'TECNOLOGIA (TEC)', 1.55);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (80,  
'VENDAS (VEN)', 1.87);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (90,  
'EXECUTIVO (EXEC)', 1.92);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (110,  
'CONTABILIDADE (CONTAB)', 2.35);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (190,  
'CONTRATOS (CTT)', 2.23);
```

Inserir dados na tabela Empregados: Número, Nome, Departamento; 100, Antônio Palmeira, 10; 101, José Figueira, 20; 102, Maria Manguieira, 10; 103,

João Abacateiro, 50; 104, Carlos Castanheiro, 60; 107, Mônica Videira, 90; 124, Pedro Laranjeira, 80; 141, Tiago Pessegueiro, 110; 142, Renata Videira, 80; 143, Gabriel Macieira, 110;

--Todos

```
INSERT INTO EMPREGADO VALUES (100, 'ANTÔNIO PALMEIRA', 10);
INSERT INTO EMPREGADO VALUES (101, 'JOSÉ FIGUEIRA', 20);
INSERT INTO EMPREGADO VALUES (102, 'MARIA MANGUEIRA', 10);
INSERT INTO EMPREGADO VALUES (103, 'JOÃO ABACATEIRO', 50);
INSERT INTO EMPREGADO VALUES (104, 'CARLOS CASTANHEIRO', 60);
INSERT INTO EMPREGADO VALUES (107, 'MÔNICA VIDEIRA', 90);
INSERT INTO EMPREGADO VALUES (124, 'PEDRO LARANJEIRA', 80);
INSERT INTO EMPREGADO VALUES (141, 'TIAGO PESSEGUIERO', 110);
INSERT INTO EMPREGADO VALUES (142, 'RENATA VIDEIRA', 80);
INSERT INTO EMPREGADO VALUES (143, 'GABRIEL MACIEIRA', 110);
```

2.1.3 Alteração de Metadados (Modificar Estruturas)

Comando que permite modificar as estruturas de armazenamento.

2.1.3.1 Sintaxe

ALTER TABLE nome_tbl especificação_alter [, especificação_alter ...]

especificação_alter:

```
  ADD [COLUMN] definição_create [FIRST | AFTER nome_coluna ]
  | ADD [COLUMN] (definição_create, definição_create,...)
  | ADD INDEX [nome_indice] (index_nome_col,...)
  | ADD [CONSTRAINT [symbol]] PRIMARY KEY (index_col_name,...)
  | ADD [CONSTRAINT [symbol]] UNIQUE [index_name] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]] FOREIGN KEY [index_name]
(index_col_name,...)
  [definição_referencia]
  | ALTER [COLUMN] nome_col {SET DEFAULT literal | DROP DEFAULT}
  | CHANGE [COLUMN] nome_col_antigo definição_create
    [FIRST | AFTER nome_coluna]
  | MODIFY [COLUMN] definição_create [FIRST | AFTER nome_coluna]
  | DROP [COLUMN] nome_col
  | DROP PRIMARY KEY
  | DROP INDEX nome_indice
  | RENAME [TO] nome_nova_tbl
  | ORDER BY col
```

2.1.3.2 Práticas

Alterar os dados da tabela Empregado adicionando as colunas: Primeironome, literal variável (25); Sobrenome, literal variável (25); Departamento, (FK) número inteiro; Cargo, (FK) número inteiro, atualização cascata; Salário, número real;

Comissão, número inteiro; Nascimento, data; Apelido, literal fixo(10), não nulo; Gerente, número inteiro:

```
--SQLServer
ALTER TABLE EMPREGADO
ADD PRIMEIRONOME VARCHAR(25),
SOBRENOME VARCHAR(25),
CARGO INTEGER,
SALARIO REAL,
COMISSAO INTEGER,
NASCIMENTO DATE,
APELIDO VARCHAR(10),
GERENTE INTEGER;
```

```
--Postgre | MySQL | Firebird
ALTER TABLE EMPREGADO
ADD PRIMEIRONOME VARCHAR(25),
ADD SOBRENOME VARCHAR(25),
ADD CARGO INTEGER,
ADD SALARIO REAL,
ADD COMISSAO INTEGER,
ADD NASCIMENTO DATE,
ADD APELIDO VARCHAR(10),
ADD GERENTE INTEGER;
```

```
--Oracle
ALTER TABLE EMPREGADO
ADD (PRIMEIRONOME VARCHAR(25),
SOBRENOME VARCHAR(25),
CARGO INTEGER,
SALARIO REAL,
COMISSAO INTEGER,
NASCIMENTO DATE,
APELIDO VARCHAR(10),
GERENTE INTEGER);
```

```
--SQLServer
ALTER TABLE EMPREGADO
ADD FOREIGN KEY (CARGO) REFERENCES CARGO (NUMERO) ON
UPDATE CASCADE,
FOREIGN KEY (GERENTE) REFERENCES EMPREGADO (NUMERO);
```

```
--Postgre | MySQL | Firebird
ALTER TABLE EMPREGADO
```

```
ADD FOREIGN KEY (CARGO) REFERENCES CARGO (NUMERO) ON  
UPDATE CASCADE,  
ADD FOREIGN KEY (GERENTE) REFERENCES EMPREGADO (NUMERO);
```

--Oracle

```
ALTER TABLE EMPREGADO  
ADD FOREIGN KEY (CARGO) REFERENCES CARGO;  
ALTER TABLE EMPREGADO  
ADD FOREIGN KEY (GERENTE) REFERENCES EMPREGADO;
```

Fazer a alteração da tabela Empregado:

--SQLServer | Postgre | Oracle | MySQL

```
ALTER TABLE EMPREGADO  
DROP COLUMN NOME;
```

--Firebird

```
ALTER TABLE EMPREGADO  
DROP NOME;
```

2.1.4 Alteração de Dados (Atualização)

```
UPDATE nome_tabela  
    SET nome_coluna1=expr1 [, nome_coluna2=expr2 ...]  
    [WHERE definição]  
    [ORDER BY ...]
```

2.1.4.1 Dicas

Pode-se alterar linha;

Pode-se alterar listas;

Pode-se alterar subconsultas;

Pode-se alterar linhas com base em outra tabela.

2.1.4.2 Práticas

Atualizar as linhas de Empregado com os valores: Número, Primeironome, Sobrenome, Cargo, Departamento, Gerente, Salário, Comissão, Nascimento, Apelido; 100, Antônio, Palmeira, 1, 10, nulo, 1000, nulo, 01/07/80, Tonho; 101, José, Figueira, 1, 20, 100, 1500, nulo, 18/05/70, Zé; 102, Maria, Mangueira, 2, 10, 100, 2000, 3, 22/07/80, Ma; 103, João, Abacateiro, 3, 50, 100, 1500, 2, 03/03/71, Jo; 104, Carlos, Castanheiro, nulo, 60, 100, 2500, 3, 01/01/65, Ca; 107, Mônica, Videira, 1, 90, nulo, 1000, 2, 15/02/60, Mo; 124, Pedro, Laranjeira, 2, 80, 107, 1100, nulo, 30/07/83, Pedraca; 141, Tiago, Pessegueiro, 3, 80, 107, 2000, 2, 21/06/78, Ti; 142, Renata, Videira, 4, 80, 107, 1500, nulo, 02/12/72, Rê; 143, Gabriel, Macieira, nulo, 110, 107, 2000, 2, 11/11/81, Biel;

--SQLServer | Postgre | Oracle

```
UPDATE EMPREGADO SET
CARGO=1, GERENTE=NULL, SALARIO=1000, COMISSAO=NULL,
NASCIMENTO='01/07/80', APELIDO='TONHO' WHERE NUMERO=100;
UPDATE EMPREGADO SET
CARGO=1, GERENTE=100, SALARIO=1500, COMISSAO=NULL,
NASCIMENTO='18/05/70', APELIDO='ZÉ' WHERE NUMERO=101;
UPDATE EMPREGADO SET
CARGO=2, GERENTE=100, SALARIO=2000, COMISSAO=3,
NASCIMENTO='22/07/80', APELIDO='MA' WHERE NUMERO=102;
UPDATE EMPREGADO SET
CARGO=3, GERENTE=100, SALARIO=1500, COMISSAO=2,
NASCIMENTO='03/03/71', APELIDO='JO' WHERE NUMERO=103;
UPDATE EMPREGADO SET
CARGO=NULL, GERENTE=100, SALARIO=2500, COMISSAO=3,
NASCIMENTO='01/01/65', APELIDO='CA' WHERE NUMERO=104;
UPDATE EMPREGADO SET
CARGO=1, GERENTE=NULL, SALARIO=1000, COMISSAO=2,
NASCIMENTO='15/02/60', APELIDO='MO' WHERE NUMERO=107;
UPDATE EMPREGADO SET
CARGO=2, GERENTE=107, SALARIO=1100, COMISSAO=NULL,
NASCIMENTO='30/07/83', APELIDO='PEDROCA' WHERE NUMERO=124;
UPDATE EMPREGADO SET
CARGO=3, GERENTE=107, SALARIO=2000, COMISSAO=2,
NASCIMENTO='21/06/78', APELIDO='TI' WHERE NUMERO=141;
UPDATE EMPREGADO SET
CARGO=4, GERENTE=107, SALARIO=1500, COMISSAO=NULL,
NASCIMENTO='02/12/72', APELIDO='RE' WHERE NUMERO=142;
UPDATE EMPREGADO SET
CARGO=NULL, GERENTE=107, SALARIO=2000, COMISSAO=2,
NASCIMENTO='11/11/81', APELIDO='BIEL' WHERE NUMERO=143;
```

--MySQL

```
UPDATE EMPREGADO SET
CARGO=1, GERENTE=NULL, SALARIO=1000, COMISSAO=NULL,
NASCIMENTO='80/07/01', APELIDO='TONHO' WHERE NUMERO=100;
UPDATE EMPREGADO SET
CARGO=1, GERENTE=100, SALARIO=1500, COMISSAO=NULL,
NASCIMENTO='70/05/18', APELIDO='ZÉ' WHERE NUMERO=101;
UPDATE EMPREGADO SET
CARGO=2, GERENTE=100, SALARIO=2000, COMISSAO=3,
NASCIMENTO='80/07/22', APELIDO='MA' WHERE NUMERO=102;
UPDATE EMPREGADO SET
CARGO=3, GERENTE=100, SALARIO=1500, COMISSAO=2,
NASCIMENTO='71/03/03', APELIDO='JO' WHERE NUMERO=103;
UPDATE EMPREGADO SET
```

```
CARGO=NULL, GERENTE=100, SALARIO=2500, COMISSAO=3,  
NASCIMENTO='65/01/01', APELIDO='CA' WHERE NUMERO=104;  
UPDATE EMPREGADO SET  
CARGO=1, GERENTE=NULL, SALARIO=1000, COMISSAO=2,  
NASCIMENTO='60/02/15', APELIDO='MO' WHERE NUMERO=107;  
UPDATE EMPREGADO SET  
CARGO=2, GERENTE=107, SALARIO=1100, COMISSAO=NULL,  
NASCIMENTO='83/07/30', APELIDO='PEDROCA' WHERE NUMERO=124;  
UPDATE EMPREGADO SET  
CARGO=3, GERENTE=107, SALARIO=2000, COMISSAO=2,  
NASCIMENTO='78/06/21', APELIDO='TI' WHERE NUMERO=141;  
UPDATE EMPREGADO SET  
CARGO=4, GERENTE=107, SALARIO=1500, COMISSAO=NULL,  
NASCIMENTO='72/12/02', APELIDO='RE' WHERE NUMERO=142;  
UPDATE EMPREGADO SET  
CARGO=NULL, GERENTE=107, SALARIO=2000, COMISSAO=2,  
NASCIMENTO='81/11/11', APELIDO='BIEL' WHERE NUMERO=143;
```

--Firebird

```
UPDATE EMPREGADO SET  
CARGO=1, GERENTE=NULL, SALARIO=1000, COMISSAO=NULL,  
NASCIMENTO='07/01/80', APELIDO='TONHO' WHERE NUMERO=100;  
UPDATE EMPREGADO SET  
CARGO=1, GERENTE=100, SALARIO=1500, COMISSAO=NULL,  
NASCIMENTO='05/18/70', APELIDO='ZÉ' WHERE NUMERO=101;  
UPDATE EMPREGADO SET  
CARGO=2, GERENTE=100, SALARIO=2000, COMISSAO=3,  
NASCIMENTO='07/22/80', APELIDO='MA' WHERE NUMERO=102;  
UPDATE EMPREGADO SET  
CARGO=3, GERENTE=100, SALARIO=1500, COMISSAO=2,  
NASCIMENTO='03/03/71', APELIDO='JO' WHERE NUMERO=103;  
UPDATE EMPREGADO SET  
CARGO=NULL, GERENTE=100, SALARIO=2500, COMISSAO=3,  
NASCIMENTO='01/01/65', APELIDO='CA' WHERE NUMERO=104;  
UPDATE EMPREGADO SET  
CARGO=1, GERENTE=NULL, SALARIO=1000, COMISSAO=2,  
NASCIMENTO='02/15/60', APELIDO='MO' WHERE NUMERO=107;  
UPDATE EMPREGADO SET  
CARGO=2, GERENTE=107, SALARIO=1100, COMISSAO=NULL,  
NASCIMENTO='07/30/83', APELIDO='PEDROCA' WHERE NUMERO=124;  
UPDATE EMPREGADO SET  
CARGO=3, GERENTE=107, SALARIO=2000, COMISSAO=2,  
NASCIMENTO='06/21/78', APELIDO='TI' WHERE NUMERO=141;  
UPDATE EMPREGADO SET  
CARGO=4, GERENTE=107, SALARIO=1500, COMISSAO=NULL,
```

```
NASCIMENTO='12/02/72', APELIDO='RE' WHERE NUMERO=142;  
UPDATE EMPREGADO SET  
CARGO=NULL, GERENTE=107, SALARIO=2000, COMISSAO=2,  
NASCIMENTO='11/11/81', APELIDO='BIEL' WHERE NUMERO=143;
```

2.1.5 Algumas Funções

Permite aumentar a capacidade de execução de consultas dos bancos de dados.

2.1.5.1 Sintaxe

SQLServer | Postgre | MySQL | Firebird

SUBSTRING(*String* texto, posicao_inicial, tamanho) - retorna uma *String* com o comprimento definido em "tamanho" extraída da *String* "texto", a partir da "posicao_inicial";
Oracle: SUBSTR

SQLServer

CHARINDEX(*String* pesquisado, campo, posicao_inicial) – retorna a posição do *String* pesquisado dentro do campo informado.

Postgre: STRPOS

Oracle | MySQL: INSTR

Firebird: POSITION

2.1.5.2 Práticas

Fazendo uso das funções *subString* e *charindex* separar o conteúdo do campo Nome nos campos *Primeironome* e *Sobrenome*. Depois apagar o campo Nome ficando: Número, *Primeironome*, *Sobrenome*, Departamento; 100, Antônio, Palmeira, 10; 101, José, Figueira, 20; 102, Maria, Mangueira, 10; 103, João, Abacateiro, 50; 104, Carlos, Castanheiro, 60; 107, Mônica, Videira, 90; 124, Pedro, Laranjeira, 80; 141, Tiago, Pessegueiro, 80; 142, Renata, Videira, 80; 143, Gabriel, Macieira, 110:

--SQLServer

```
UPDATE EMPREGADO SET
```

```
PRIMEIRONOME = SUBSTRING(NOME,1,CHARINDEX(' ',NOME)-1),
```

```
SOBRENOME = SUBSTRING(NOME,CHARINDEX(' ',nome)+1,100);
```

--Postgre

```
UPDATE EMPREGADO SET
```

```
PRIMEIRONOME = SUBSTRING(NOME,1,STRPOS(NOME, ' ')-1),
```

```
SOBRENOME = SUBSTRING(NOME,STRPOS(NOME, ' ')+1,100);
```

--Oracle

```
UPDATE EMPREGADO SET  
PRIMEIRONOME = SUBSTR(NOME,1,INSTR(NOME, ' ')-1),  
SOBRENOME = SUBSTR(NOME,INSTR(NOME, ' ')+1,100);
```

--MySQL

```
UPDATE EMPREGADO SET  
PRIMEIRONOME = SUBSTRING(NOME,1,INSTR(NOME, ' ')-1),  
SOBRENOME = SUBSTRING(NOME,INSTR(NOME, ' ')+1,100);
```

--Firebird

```
UPDATE EMPREGADO SET  
PRIMEIRONOME = SUBSTRING(NOME FROM 1 FOR POSITION(' ', NOME)-  
1),  
SOBRENOME = SUBSTRING(NOME FROM POSITION(' ', NOME)+1 FOR  
100);
```

2.1.6 Índices

Permite criar estruturas de indexação que auxiliam as estruturas de armazenamento aumentando a performance no acesso aos dados.

2.1.6.1 Sintaxe

```
CREATE [UNIQUE] INDEX nome_indice  
ON nome_tabela (index_col_name,...)  
index_col_name:  
col_name [(length)] [ASC | DESC]
```

Trata-se de uma referência associada a uma chave, que tem a finalidade de otimizar a pesquisa, permitindo assim a localização rápida de um registro quando efetuada uma consulta. Trata-se de uma estrutura auxiliar associado a uma tabela, que possui ponteiros para os dados armazenados em colunas específicas. Fazendo uma analogia, o banco de dados usa o índice de maneira semelhante ao índice remissivo de um livro, verificando um determinado assunto no índice e depois localizando a sua posição.

2.1.6.2 Práticas

Criar índices por: Primeironome e Sobrenome; Sobrenome; Salário; Comissão; Cargo; Departamento; Apelido (Único):

```
CREATE INDEX EMPREGADOIX1 ON EMPREGADO (PRIMEIRONOME);  
CREATE INDEX EMPREGADOIX2 ON EMPREGADO (SOBRENOME);  
CREATE INDEX EMPREGADOIX3 ON EMPREGADO (SALARIO);  
CREATE INDEX EMPREGADOIX4 ON EMPREGADO (COMISSAO);  
CREATE INDEX EMPREGADOIX5 ON EMPREGADO (CARGO);  
CREATE INDEX EMPREGADOIX6 ON EMPREGADO (DEPARTAMENTO);  
CREATE UNIQUE INDEX EMPREGADOIX7 ON EMPREGADO (APELIDO);
```

2.1.7 Dicionário de Dados

Trata-se de uma coleção de metadados que contém definições e representações de elementos de dados. Os Sistemas Gerenciadores de Bancos de Dados (SGBD's) fazem uso dos dicionários de dados, permitindo apenas consulta por parte dos usuários, e possuem as seguintes informações: Definição dos elementos de dados; Perfis de usuários, papéis e privilégios; Descrição de objetos; Integridade de restrições; Informação de verificação; e Alocações de espaço.

2.1.7.1 Práticas

Visualizar a estrutura das tabelas Empregados e Departamentos:

--SQLServer

```
SELECT * FROM SYSOBJECTS O INNER JOIN SYSCOLUMNS C ON  
O.ID=C.ID WHERE O.NAME = 'DEPARTAMENTO';
```

--Postgre

```
SELECT  
a.attnum,  
a.attname AS field,  
t.typname AS type,  
a.attlen AS length,  
a.atttypmod AS lengthvar,  
a.attnotnull AS notnull,  
c.relname AS name  
FROM  
pg_class c,  
pg_attribute a,  
pg_type t  
WHERE  
c.relname = 'departamento' and  
a.attnum > 0  
and a.attrelid = c.oid  
and a.atttypid = t.oid  
ORDER BY c.relname
```

--Oracle | MySQL

```
DESCRIBE DEPARTAMENTO;
```

2.1.8 Consulta de Dados

Comando que permite extrair os dados das estruturas de armazenamento como desejado pelo usuários.

2.1.8.1 Sintaxe

```
SELECT
```

```
[ ALL | DISTINCT ]
[ TOP n ]
[ table_name. | table_alias. | view_name. ]
*
| column_name [ [ AS ] column_alias ]
| expression [ [ AS ] column_alias ]
[ ..., n ]
[ INTO new_table ]
FROM table
[ AS table_alias ]
[ ( column_alias_1, [ column_alias_2, ..., n ] ) ]
|
view_name [ AS table_alias ]
[ INNER JOIN | LEFT [ OUTER ] JOIN | RIGHT [ OUTER ] JOIN table [ ON
search_conditions ]
|
[ WHERE search_conditions
|
WHERE column_name =* column_name
]
[ GROUP BY { group_by_expression, [ ..., n ] } ]
[ HAVING search_conditions ]
[ UNION query ]
[ ORDER BY { order_by_expression [ ASC | DESC ] , [ ..., n ] } ]
```

2.1.8.2 Dicas

Operador de concatenação (||);

Em alguns BD se usa (+) para concatenação;

Concatena colunas ou *Strings* de caracteres a outras colunas.

2.1.8.3 Práticas

Visualizar os campos: Número e Nome (Primeironome || ' ' || Sobrenome):

--SQLServer

```
SELECT NUMERO, PRIMEIRONOME + ' ' + SOBRENOME AS NOME FROM
EMPREGADO;
```

--Postgre | Oracle | Firebird

```
SELECT NUMERO, PRIMEIRONOME || ' ' || SOBRENOME AS NOME FROM
EMPREGADO;
```

--MySQL

```
SELECT NUMERO, CONCAT(PRIMEIRONOME, ' ', SOBRENOME) AS NOME
FROM EMPREGADO;
```

Visualizar os registros dos empregados do departamento 20 ou 80 dos campos: Número; Primeironome; Sobrenome; e Departamento:

```
SELECT NUMERO, PRIMEIRONOME, SOBRENOME, DEPARTAMENTO  
FROM EMPREGADO  
WHERE DEPARTAMENTO = 20 OR DEPARTAMENTO = 80;
```

Visualizar os registros dos empregados juntamente com o nome do departamento: Número (Empregados); Primeironome (Empregados); Sobrenome (Empregados); Departamento (Empregados); e Nome (Departamentos).

```
SELECT      E.NUMERO,          E.PRIMEIRONOME,          E.SOBRENOME,  
E.DEPARTAMENTO,  
D.NOME  
FROM EMPREGADO E INNER JOIN DEPARTAMENTO D  
ON E.DEPARTAMENTO = D.NUMERO;
```

2.1.9 Precedência

Use parênteses (...) para forçar a ordem em que as expressões serão avaliadas.

2.1.10 Comparação

Se um ou ambos os argumentos são NULL, o resultado da comparação é NULL; Se ambos os argumentos em uma comparação são *Strings*, eles são comparados como *Strings*; Se ambos os argumentos são inteiros, eles são comparados como inteiros; Valores hexadecimais são tratados como *Strings* binárias se não comparadas a um número; Se um dos argumentos é uma coluna *TIMESTAMP* ou *DATETIME* e o outro argumento é uma constante, a constante é convertida para um timestamp antes da comparação ser realizada; Por padrão, comparações de *String* são feita de modo independente do caso, usando o conjunto de caracteres atual (ISO-8859-1 Latin1 por padrão); Comparação *Strings* com distinção de letras maiúsculas e minúsculas pode ser realizado com qualquer dos operadores, exceto o *LIKE*; Espaços em branco no fim da *String*, espaços, tabs e quebra de linha serão ignorados.

2.1.11 Projeção

Consiste na escolha das colunas que serão exibidas.

2.1.12 Seleção

Consiste na escolha das linhas que serão exibidas.

2.1.13 Junção

Consiste em relacionar dados de duas ou mais tabelas ou mesmo relacionar uma tabela com ela mesma podendo ser uma correlação ou autojunção.

2.1.14 Consulta Básica

Consiste em usar os argumentos básicos do comando Select que são o SELECT e FROM.

2.1.14.1 Dicas

As instruções SQL não fazem distinção entre maiúsculas e minúsculas;
As instruções SQL podem estar em uma ou mais linhas;
As palavras-chave não podem ser abreviadas ou quebradas;
Normalmente as cláusulas são colocadas em linhas separadas;
Os recuos são utilizados para dar mais legibilidade.

2.1.14.2 Práticas

Visualizar todos os registros e todos os campos da tabela departamentos.

```
SELECT * FROM DEPARTAMENTO;
```

2.1.15 Apelido (Alias)

Consiste em atribuir outro nome a um metadado mas que não irá afetar a estrutura física é apenas uma alteração lógica.

2.1.15.1 Práticas

Visualizar todos os registros nos campos Número, Nome (Primeironome + ' ' + Sobrenome) e após a consulta ser realizada observar o nome do campo para a expressão Nome. Finalize repetindo a consulta colocando o apelido Nome do Funcionário.

--SQLServer

```
SELECT NUMERO, PRIMEIRONOME + ' ' + SOBRENOME FROM  
EMPREGADO;
```

```
SELECT NUMERO, PRIMEIRONOME + ' ' + SOBRENOME AS NOME FROM  
EMPREGADO;
```

--Postgre | Oracle | Firebird

```
SELECT NUMERO, PRIMEIRONOME || ' ' || SOBRENOME FROM  
EMPREGADO;
```

```
SELECT NUMERO, PRIMEIRONOME || ' ' || SOBRENOME AS NOME FROM  
EMPREGADO;
```

--MySQL

```
SELECT NUMERO, CONCAT(PRIMEIRONOME, ' ', SOBRENOME) FROM  
EMPREGADO;
```

```
SELECT NUMERO, CONCAT(PRIMEIRONOME, ' ', SOBRENOME) AS NOME  
FROM EMPREGADO;
```

2.1.16 Campos Calculados

São campos lógicos baseados em operações aritméticas fazendo uso dos operadores e funções.

Operadores aritméticos:

- Adicionar: +
- Subtrair: -
- Multiplicar: *
- Dividir: / “retorna o quociente”

Funções matemáticas:

- Potenciação: POWER(número, potência)
- Radiciação: SQRT(número)
- Divisão: % | MOD “retorna o resto”

Precedência de Operadores:

- A multiplicação e a divisão têm prioridade sobre a adição e subtração;
- Os operadores com a mesma prioridade são avaliados da esquerda para a direita;
- Os parênteses são usados para forçar a avaliação priorizada e para esclarecer as instruções.

2.1.16.1 Práticas

Visualizar o Sobrenome, o salário, e o líquido dos empregados, considerando que todos possuem um desconto de 10% de INSS.

```
SELECT SOBRENOME, SALARIO, SALARIO - (SALARIO*0.10) AS LIQUIDO  
FROM EMPREGADO;
```

2.1.17 Valores Nulos

São campos que não possuem nenhum tipo de dado o que chamamos de valor nulo.

2.1.17.1 Dicas

Nulo é um valor que não está disponível;
Nulo é um valor desconhecido;
Nulo não é a mesma coisa que zero;
Nulo não é a mesma coisa que espaço em branco;
Para Oracle usar NVL(campo, valor);
Para MySQL usar IFNULL(campo, valor);
Para SQLServer usar ISNULL(campo, valor);
Para Postgre | Firebird usar COALESCE(campo,0).

2.1.17.2 Práticas

Visualizar o Sobrenome e o Salário Bruto (Salário + Comissão). Obs: A Comissão representa um percentual aplicado sobre o salário.

```
SELECT SOBRENOME, SALARIO + (SALARIO * COMISSAO / 100) AS BRUTO  
FROM EMPREGADO;
```

Visualizar o Sobrenome e o Salário Bruto (Salário + Comissão). Obs: Quando a Comissão for nula deve-se substituir por zero.

--SQLServer

```
SELECT SOBRENOME, SALARIO + (SALARIO * ISNULL(COMISSAO,0) / 100)  
AS BRUTO FROM EMPREGADO;
```

--Postgre | Firebird

```
SELECT SOBRENOME, SALARIO + (SALARIO * COALESCE(COMISSAO,0) /  
100) AS BRUTO FROM EMPREGADO;
```

--Oracle

```
SELECT SOBRENOME, SALARIO + (SALARIO * NVL(COMISSAO,0) / 100) AS  
BRUTO FROM EMPREGADO;
```

--MySQL

```
SELECT SOBRENOME, SALARIO + (SALARIO * IFNULL(COMISSAO,0) / 100)  
AS BRUTO FROM EMPREGADO;
```

2.1.18 Duplicidade

Consiste em eliminar na consulta, isto é, apenas na visualização dos dados, as duplicidades ou valores repetidos.

2.1.18.1 Dicas

A exibição default das consultas é de todas as linhas, incluindo linhas duplicadas.

2.1.18.2 Práticas

Visualizar o Sobrenome dos Empregados.

```
SELECT SOBRENOME FROM EMPREGADO;
```

Visualizar o Sobrenome dos Empregados sem duplicidade.

```
SELECT DISTINCT SOBRENOME FROM EMPREGADO;
```

2.1.19 Condicionamento

São seleções de dados resultantes de condicionamentos lógicos realizados pela cláusula WHERE.

Operadores de Comparação:

- Igual a: =
- Maior que: >
- Maior que ou igual a: >=
- Menor que: <
- Menor que ou igual a: <=
- Diferente de: <>

Operadores Lógicos:

- Verdadeiro para ambas expressões verdadeiras: AND
- Verdadeiro bastando uma das expressões verdadeiras: OR
- Inverso da expressão: NOT

Regras de Precedência

- Operadores Aritméticos;
- Operador de Concatenação;
- Condições de Comparação;
- IS [NOT] NULL, [NOT] LIKE, [NOT] IN;
- [NOT] BETWEEN;
- Condição lógica NOT;
- Condição lógica AND;
- Condição lógica OR.

2.1.19.1 Dicas

As *Strings* de caracteres e valores de data aparecem entre aspas simples;

Os valores de caractere fazem distinção entre maiúsculas e minúsculas;

Os valores de data fazem distinção entre formatos;

O formato de data default é DD-MON-YY.

2.1.19.2 Práticas

Visualizar o Número e Sobrenome de todos os empregados do departamento 90.

```
SELECT NUMERO, SOBRENOME  
FROM EMPREGADO  
WHERE DEPARTAMENTO = 90;
```

Visualizar o Número, Sobrenome e Nascimento de todos os empregados que nasceram em 22/07/80.

--SQLServer | postgre | Oracle

```
SELECT NUMERO, SOBRENOME, NASCIMENTO FROM EMPREGADO  
WHERE NASCIMENTO = '22/07/80';
```

--MySQL

```
SELECT NUMERO, SOBRENOME, NASCIMENTO FROM EMPREGADO  
WHERE NASCIMENTO = '80/07/22';
```

--Firebird

```
SELECT NUMERO, SOBRENOME, NASCIMENTO FROM EMPREGADO  
WHERE NASCIMENTO = '07/22/80';
```

Visualizar o Número, Sobrenome e Nascimento de todos os empregados que nasceram entre (inclusive) 01/01/70 e 31/12/80.

--SQLServer | postgre | Oracle

```
SELECT NUMERO, SOBRENOME, NASCIMENTO FROM EMPREGADO  
WHERE NASCIMENTO  
BETWEEN '01/01/70' AND '31/12/80';
```

--MySQL

```
SELECT NUMERO, SOBRENOME, NASCIMENTO FROM EMPREGADO  
WHERE NASCIMENTO  
BETWEEN '70/01/01' AND '80/12/31';
```

--Firebird

```
SELECT NUMERO, SOBRENOME, NASCIMENTO FROM EMPREGADO  
WHERE NASCIMENTO  
BETWEEN '01/01/70' AND '12/31/80';
```

Visualizar o Número, Sobrenome e Departamento de todos os empregados que pertencem aos departamentos: 10, 20, 50, 60, 80 ou 90.

```
SELECT NUMERO, SOBRENOME, DEPARTAMENTO  
FROM EMPREGADO  
WHERE DEPARTAMENTO IN (10, 20, 50, 60, 80, 90);
```

Visualizar o Número e Sobrenome de todos os empregados que possuem 'eira' no sobrenome.

```
SELECT NUMERO, SOBRENOME  
FROM EMPREGADO
```

```
WHERE SOBRENOME LIKE '%EIRA%';
```

Visualizar o Número e Sobrenome de todos os empregados que não possuem comissão.

```
SELECT NUMERO, SOBRENOME  
FROM EMPREGADO  
WHERE COMISSAO IS NULL;
```

Visualizar o Número, Sobrenome e Salário de todos os empregados que possuem salário entre (inclusive) 1500 e 2000, dos departamentos 20, 50 e 80.

```
SELECT NUMERO, SOBRENOME, SALARIO, DEPARTAMENTO  
FROM EMPREGADO  
WHERE SALARIO BETWEEN 1500 AND 2000  
AND DEPARTAMENTO IN (20, 50, 80);
```

2.1.20 Ordenação

Permite realizar classificação, isto é, uma ordenação dos dados que serão exibidos.

2.1.20.1 Sintaxe

```
[ORDER BY {coluna, expr} [ASC|DESC]];
```

2.1.20.2 Dicas

É possível utilizar o alias na cláusula de ordenação;

É possível utilizar a posição da coluna na ordenação;

Pode-se classificar por colunas que não estejam no select;

Pode-se classificar por N colunas.

2.1.20.3 Práticas

Visualizar o Sobrenome e o Salário dos Empregados ordenados por Salário em ordem decrescente.

```
SELECT SOBRENOME, SALARIO  
FROM EMPREGADO  
ORDER BY SALARIO DESC;
```

2.1.21 Aninhamento

São comandos ou funções que se apresentam uma dentro da outra.

2.1.21.1 Dicas

Funções de uma única linha podem ser aninhadas em qualquer nível;

As funções aninhadas são avaliadas do nível mais interno para o mais externo.

2.1.21.2 Práticas

Exibir o tamanho do Apelido concatenado com o Sobrenome com os brancos e depois sem os brancos; Exibir o tamanho do Apelido concatenado com o Sobrenome com os brancos

--SQLServer

```
SELECT LEN(APELIDO + ' ' + SOBRENOME) FROM EMPREGADO;
```

--Postgre | Oracle

```
SELECT LENGTH(APELIDO || ' ' || SOBRENOME) FROM EMPREGADO;
```

--MySQL

```
SELECT LENGTH(CONCAT(APELIDO, ' ', SOBRENOME)) FROM EMPREGADO;
```

--Firebird

```
SELECT CHAR_LENGTH(APELIDO || ' ' || SOBRENOME) FROM EMPREGADO;
```

Exibir o tamanho do Apelido concatenado com o Sobrenome sem os brancos.

--SQLServer

```
SELECT LEN(RTRIM(APELIDO) + ' ' + SOBRENOME) FROM EMPREGADO;
```

--Postgre | Oracle

```
SELECT LENGTH(RTRIM(APELIDO) || ' ' || SOBRENOME) FROM EMPREGADO;
```

--MySQL

```
SELECT LENGTH(CONCAT(RTRIM(APELIDO), ' ', SOBRENOME)) FROM EMPREGADO;
```

--Firebird

```
SELECT CHAR_LENGTH(TRIM(APELIDO) || ' ' || SOBRENOME) FROM EMPREGADO;
```

2.1.22 Funções

Funções são recursos avançados do SQL e podem ser utilizadas para efetuar o seguinte: Executar cálculos; Modificar itens de dados individuais; Manipular a saída para grupos de linhas; Formatar datas e números para exibição; Converter tipos de dados de coluna; Algumas funções utilizam argumentos; Sempre retornam um valor.

2.1.22.1 Tipos

Funções de uma Única Linha - operam somente linhas únicas e retornam um resultado por linha;

Funções de Várias Linhas - podem manipular grupos de linhas para fornecer um resultado e são mais conhecidas como funções de grupo.

2.1.22.2 Dicas

Servem para manipular itens de dados;

Aceitam um ou mais argumentos;

Retornam um valor para cada linha retornada;

São do tipo: Caractere, Numéricas, Data, Conversão e Gerais;

Podem ser usadas em cláusulas: SELECT, WHERE e ORDER BY.

2.1.23 Funções Caractere

LOWER – converte para minúsculas;

UPPER – converte para maiúsculas;

SUBSTRING – retorna o subString de uma String;

LENGTH – retorna o número de caracteres;

CHARINDEX – retorna a posição de um String;

REPLACE - substitui uma String por outra String;

RTRIM | LTRIM | TRIM – retira brancos.

2.1.23.1 Práticas

Exibir todos os Primeironomes em minúsculas.

```
SELECT LOWER(PRIMEIRONOME) AS PRIMEIRONOME FROM EMPREGADO;
```

Exibir todos os Sobrenomes em maiúsculas.

```
SELECT UPPER(SOBRENOME) AS PRIMEIRONOME FROM EMPREGADO;
```

Exibir os 3 primeiros caracteres do Sobrenome.

--SQLServer | Postgre | MySQL

```
SELECT SUBSTRING(SOBRENOME,1,3) FROM EMPREGADO;
```

--Oracle

```
SELECT SUBSTR(SOBRENOME,1,3) FROM EMPREGADO;
```

--Firebird

```
SELECT SUBSTRING(SOBRENOME FROM 1 FOR 3) FROM EMPREGADO;
```

Exibir o tamanho do Sobrenome.

```
--SQLServer  
SELECT LEN(SOBRENOME) FROM EMPREGADO;
```

```
--Postgre | Oracle | MySQL  
SELECT LENGTH(SOBRENOME) FROM EMPREGADO;
```

```
--Firebird  
SELECT CHAR_LENGTH(SOBRENOME) FROM EMPREGADO;
```

Exibir a posição no '(' no nome do departamento.

```
--SQLServer  
SELECT CHARINDEX('(', NOME) FROM DEPARTAMENTO;
```

```
--Postgre  
SELECT STRPOS(NOME, '(') FROM DEPARTAMENTO;
```

```
--Oracle | MySQL  
SELECT INSTR(NOME, '(') FROM DEPARTAMENTO;
```

```
--Firebird  
SELECT POSITION('(' IN NOME) FROM DEPARTAMENTO;
```

2.1.24 Funções Numérica

ROUND – arredonda a coluna;

TRUNC – trunca a coluna;

2.1.24.1 Práticas

Exibir o Fator.

```
SELECT FATOR FROM DEPARTAMENTO;
```

Exibir o Fator arredondando para 1 decimal.

```
SELECT ROUND(FATOR,1) FROM DEPARTAMENTO;
```

Exibir o Fator truncando para 1 decimal.

```
--SQLServer  
--SELECT ROUND(FATOR, 1, 2) FROM DEPARTAMENTO;
```

```
--Postgre | Oracle | Firebird  
SELECT TRUNC(FATOR, 1) FROM DEPARTAMENTO;
```

```
--MySQL
```

```
SELECT TRUNCATE(FATOR, 1) FROM DEPARTAMENTO;
```

2.1.25 MOD | %

Retorna o resto da divisão.

2.1.25.1 Dicas

Para exibir dados que não estão inseridos em tabelas do banco de dados pode-se fazer uso de uma tabela fictícia usada para exibir resultados chamada de DUAL.

2.1.25.2 Práticas

Visualizar o quociente da divisão de 5 por 2.

```
--SQLServer | Firebird  
SELECT 5 / 2 FROM TMP;
```

```
--Postgre | MySQL  
SELECT 5 / 2;
```

```
--Oracle  
SELECT 5 / 2 FROM DUAL;
```

Visualizar o resto da divisão de 5 por 2.

```
--SQLServer  
SELECT 5 % 2 FROM TMP;
```

```
--Postgre | MySQL  
SELECT 5 MOD 2;
```

```
--Oracle  
SELECT MOD (5, 2) FROM DUAL;
```

```
--Firebird  
SELECT MOD(5, 2) FROM TMP;
```

2.1.26 Funções Data

Retornam dados no formato de datas.

2.1.26.1 Sintaxe

```
GETDATE | SYSDATE
```

2.1.26.2 Dicas

Os Bancos de Dados (BD) armazenam as datas em formato de número inteiro: século, ano, mês, dia, horas, minutos e segundos;

O formato de exibição default depende do BD, mas no geral é utilizado a exibição DD-MMM-AA;

Pode-se adicionar ou subtrair (aritmética com datas)

Operação; Resultado; Descrição

Data+Número; Data; Adiciona dias a datas

Data-Número; Data; Subtrai dias a datas

Data-Data; Dias; Subtrai uma data de outra

Data+Número/24; Data; Adiciona hora a datas

2.1.26.3 Práticas

Exiba e verifique o retorno: Exibir a data corrente; Exibir Data+Número; Exibir Data-Número; Exibir Data-Data; Exibir Data+Número/24.

--SQLServer

```
SELECT GETDATE() FROM TMP;  
SELECT GETDATE() + 1 FROM TMP;  
SELECT GETDATE() - 1 FROM TMP;  
SELECT GETDATE() - '13/11/67' FROM TMP;  
SELECT GETDATE() + 10/24 FROM TMP;  
SELECT YEAR(GETDATE()) FROM TMP;  
SELECT MONTH(GETDATE()) FROM TMP;  
SELECT DAY(GETDATE()) FROM TMP;
```

--Postgre

```
SELECT CURRENT_TIMESTAMP;  
SELECT CURRENT_TIMESTAMP + INTERVAL '1DAY';  
SELECT CURRENT_TIMESTAMP - INTERVAL '1DAY';  
SELECT CURRENT_TIMESTAMP - '13/11/67';  
SELECT CURRENT_TIMESTAMP + INTERVAL '5HOURS';  
SELECT EXTRACT(YEAR FROM CURRENT_TIMESTAMP);  
SELECT EXTRACT(MONTH FROM CURRENT_TIMESTAMP);  
SELECT EXTRACT(DAY FROM CURRENT_TIMESTAMP);
```

--Oracle

```
SELECT SYSDATE FROM DUAL;  
SELECT SYSDATE + 1 FROM DUAL;  
SELECT SYSDATE - 1 FROM DUAL;  
SELECT SYSDATE - '13/11/67' FROM DUAL;  
SELECT SYSDATE + 10/24 FROM DUAL;  
SELECT YEAR(SYSDATE) FROM DUAL;  
SELECT MONTH(SYSDATE) FROM DUAL;  
SELECT DAY(SYSDATE) FROM DUAL;
```

--MySQL

```
SELECT CURRENT_TIMESTAMP;
```

```
SELECT CURRENT_TIMESTAMP + 1;  
SELECT CURRENT_TIMESTAMP - 1;  
SELECT CURRENT_TIMESTAMP - '13/11/67';  
SELECT CURRENT_TIMESTAMP + 10/24;  
SELECT EXTRACT(YEAR FROM CURRENT_TIMESTAMP);  
SELECT EXTRACT(MONTH FROM CURRENT_TIMESTAMP);  
SELECT EXTRACT(DAY FROM CURRENT_TIMESTAMP);
```

--Firebird

```
SELECT CURRENT_TIMESTAMP FROM TMP;  
SELECT CURRENT_TIMESTAMP + 1 FROM TMP;  
SELECT CURRENT_TIMESTAMP - 1 FROM TMP;  
SELECT CURRENT_TIMESTAMP + 10/24 FROM TMP;  
SELECT EXTRACT(YEAR FROM CURRENT_TIMESTAMP) FROM TMP;  
SELECT EXTRACT(MONTH FROM CURRENT_TIMESTAMP) FROM TMP;  
SELECT EXTRACT(DAY FROM CURRENT_TIMESTAMP) FROM TMP;
```

2.1.27 Função Conversão

Conversão de dados entre diversos tipos.

2.1.27.1 Sintaxe

CAST

2.1.27.2 Práticas

Exibir a conversão do salário para *String*.

```
SELECT CAST(SALARIO AS CHAR(10)) FROM EMPREGADO;
```

2.1.28 Função de Busca e Troca

Recupera e altera dados.

2.1.28.1 Sintaxe

REPLACE

2.1.28.2 Práticas

Trocar no nome de departamento: Admin para Adm; Exec para Exe; e Contab para Con.

```
SELECT  
REPLACE(NOME, '(ADMIN)', '(ADM)'),  
REPLACE(NOME, '(EXEC)', '(EXE)'),  
REPLACE(NOME, '(CONTAB)', '(CON)')  
FROM DEPARTAMENTO;
```

2.1.29 Função de Condicionamento

Condiciona a exibição dos dados permitindo o uso de expressões condicionais.

2.1.29.1 Sintaxe

```
CASE expr WHEN cond THEN retorno  
[WHEN condN THEN retornoN]  
[ELSE retorno]  
END
```

2.1.29.2 Práticas

Visualizar Número, Sobrenome, Salário e Departamento dos empregados;
Visualizar Número, Sobrenome, Salário e Departamento dos empregados aplicando 10% de aumento nos salários dos depart. 10 e 20, 15% dos depart. 80 e 90, e 5% nos demais.

```
SELECT NUMERO, SOBRENOME,  
CASE  
WHEN DEPARTAMENTO IN (10,20) THEN SALARIO + (SALARIO*0.10)  
WHEN DEPARTAMENTO IN (80,90) THEN SALARIO + (SALARIO*0.15)  
ELSE SALARIO + (SALARIO*0.05)  
END AS SALARIO,  
DEPARTAMENTO  
FROM EMPREGADO;
```

2.1.30 Produto Cartesiano

Um produto cartesiano será formado quando uma condição de junção for omitida e como consequência todas as linhas da primeira tabela serão unidas a todas as linhas da segunda tabela.

2.1.30.1 Práticas

Visualizar Número, Sobrenome e Nome de Departamento dos empregados não colocando a junção das tabelas e observe o resultado.

--Todos

```
SELECT E.NUMERO, E.SOBRENOME, D.NOME FROM EMPREGADO E,  
DEPARTAMENTO D;
```

--SQLServer | Postgre | Oracle | MySQL

```
SELECT E.NUMERO, E.SOBRENOME, D.NOME FROM EMPREGADO E  
CROSS JOIN DEPARTAMENTO D;
```

2.1.31 Junção

Quando forem necessários dados de mais de uma tabela no banco de dados, será usada uma condição de junção. As linhas de uma tabela podem ser unidas a linhas de outra tabela de acordo com os valores comuns existentes nas

colunas correspondentes, ou seja, em geral colunas de chave primária e chave estrangeira.

2.1.31.1 Tipos de Junção

- Eqüijunção
- Não-eqüijunção
- Junção externa
- Autojunção

2.1.31.2 Dicas

Quando criar uma instrução SELECT que una tabelas, coloque o nome da tabela antes do nome da coluna para fins de clareza e para melhorar o acesso ao banco de dados;

Caso apareça o mesmo nome de coluna em mais de uma tabela, o nome da coluna deve vir antecedido do nome da tabela (AMBIGUIDADE);

Para unir N tabelas, é necessário um mínimo de N-1 condições de junção. Por exemplo, para unir 4 tabelas, é necessário um mínimo de 3 junções. Essa regra pode não se aplicar se sua tabela tiver uma chave primária concatenada. Nesse caso, será necessário mais de uma coluna para identificar exclusivamente cada linha;

Para melhorar a sintaxe deve-se apelidar as tabelas.

2.1.32 Equijunção

Junção por igualdade.

2.1.32.1 Dicas

Equijunção também são chamadas de junções simples ou internas;

Observe que nas junções simples ou internas se não existe uma relação a tupla não é gerada.

Tupla: Cada linha formada por uma lista ordenada de colunas representa um registro. Os registros não precisam conter informações em todas as colunas, podendo assumir valores nulos quando assim se fizer necessário.

2.1.32.2 Práticas

Exibir o Sobrenome do empregado e o Nome do departamento de um empregado.

```
SELECT E.SOBRENOME, D.NOME  
FROM EMPREGADO E  
INNER JOIN DEPARTAMENTO D  
ON E.DEPARTAMENTO = D.NUMERO;
```

Determine o nome do cargo e do departamento de um empregado.

```
SELECT E.SOBRENOME, C.NOME, D.NOME  
FROM EMPREGADO E  
INNER JOIN CARGO C  
ON E.CARGO = C.NUMERO  
INNER JOIN DEPARTAMENTO D  
ON E.DEPARTAMENTO = D.NUMERO;
```

2.1.33 Não-equijunção

São junções que não usam condições de igualdade.

2.1.33.1 Práticas

Exibir o Número, Sobrenome, nome do cargos de cada empregados que podem ser galgados na empresa.

```
SELECT E.NUMERO, E.SOBRENOME, C.NOME  
FROM EMPREGADO E  
INNER JOIN CARGO C  
ON C.NUMERO > E.CARGO;
```

2.1.34 Junção Externa

Quando se deseja exibir dados que não possuem uma relação comum entre tabelas.

2.1.34.1 Dicas

Em banco como Oracle pode-se utilizar a extensão (+) e em SQL Server (*);
Só pode ser utilizado de um lado da expressão;
Não se pode utilizar o operador IN e nem o OR.

2.1.34.2 Práticas

Exibir o Número, Sobrenome, Apelido, nome do cargo de todos os empregados, mesmo daqueles que ainda não possuem cargos atribuídos.

```
SELECT E.NUMERO, E.SOBRENOME, E.APELIDO, C.NOME  
FROM EMPREGADO E  
RIGHT JOIN CARGO C  
ON C.NUMERO = E.CARGO;
```

2.1.35 Autojunção

Quando se une uma tabela a ela mesma.

2.1.35.1 Práticas

Exibir o Número, Primeironome dos empregados e o Número e o Primeironome dos seus respectivos gerentes.

```
SELECT E.NUMERO, E.PRIMEIRONOME, G.NUMERO, G.PRIMEIRONOME
```

```
FROM EMPREGADO E LEFT JOIN EMPREGADO G  
ON E.GERENTE = G.NUMERO;
```

2.1.36 Funções de Grupo

Operam um conjunto de linhas, as funções de grupo operam em conjuntos de linhas para fornecer um resultado por grupo. Esses conjuntos podem ser a tabela inteira ou dividida em grupos.

2.1.36.1 Dicas

Funções de grupo são mais conhecidas como funções agregadas: AVG, COUNT, MAX, MIN e SUM;

ALL faz com que a função considere todos os valores, inclusive as duplicidades;

DISTINCT faz com que a função considere somente valores não duplicados;

Todas as funções de grupos ignoram valores nulos, para garantir a aplicabilidade para todos os registros use (*).

2.1.36.2 Práticas

Exibir a média de salários pago pelo empregador.

```
SELECT AVG(SALARIO) FROM EMPREGADO;
```

Exibir o somatório dos salários.

```
SELECT SUM(SALARIO) FROM EMPREGADO;
```

Exibir a quantidade de empregados.

```
SELECT COUNT(SALARIO) FROM EMPREGADO;
```

Exibir o maior salário.

```
SELECT MAX(SALARIO) FROM EMPREGADO;
```

Exibir o menor salário.

```
SELECT MIN(SALARIO) FROM EMPREGADO;
```

2.1.37 Grupos de Dados

Usa-se o GROUP BY para dividir as linhas de uma tabela em grupos.

2.1.37.1 Práticas

Exibir a média de salários pagos por Departamento.

```
SELECT D.NOME, AVG(E.SALARIO) AS MEDIAS
FROM DEPARTAMENTO D INNER JOIN EMPREGADO E
ON D.NUMERO = E.DEPARTAMENTO
GROUP BY D.NOME;
```

Exibir o somatório dos salários pagos por Cargo.

```
SELECT C.NOME, SUM(E.SALARIO) AS SOMATORIO
FROM CARGO C INNER JOIN EMPREGADO E
ON C.NUMERO = E.CARGO
GROUP BY C.NOME;
```

Exibir a quantidade de empregados por gestor.

```
SELECT G.SOBRENOME, COUNT(E.NUMERO) AS QTDE
FROM EMPREGADO G INNER JOIN EMPREGADO E
ON G.NUMERO = E.GERENTE
GROUP BY G.SOBRENOME;
```

Exibir o maior salário por gestor.

```
SELECT G.SOBRENOME, MAX(E.SALARIO) AS MAIORSALARIO
FROM EMPREGADO G INNER JOIN EMPREGADO E
ON G.NUMERO = E.GERENTE
GROUP BY G.SOBRENOME;
```

Exibir o menor salário por gestor.

```
SELECT G.SOBRENOME, MIN(E.SALARIO) AS MENORSALARIO
FROM EMPREGADO G INNER JOIN EMPREGADO E
ON G.NUMERO = E.GERENTE
GROUP BY G.SOBRENOME;
```

2.1.38 Grupo de Dados com Filtro

Usa-se o HAVING para filtrar as linhas de uma tabela em grupos.

2.1.38.1 Práticas

Exibir a média de salários pagos por Departamento cujo a média seja maior ou igual a 2000.

```
SELECT D.NOME, AVG(E.SALARIO) AS MEDIAS
FROM DEPARTAMENTO D INNER JOIN EMPREGADO E
ON D.NUMERO = E.DEPARTAMENTO
GROUP BY D.NOME
HAVING AVG(E.SALARIO) >= 2000;
```

Exibir o somatório dos salários pagos por Cargo cujo o somatório seja menor que 1500.

```
SELECT D.NOME, SUM(E.SALARIO) AS MEDIAS
FROM DEPARTAMENTO D INNER JOIN EMPREGADO E
ON D.NUMERO = E.DEPARTAMENTO
GROUP BY D.NOME
HAVING AVG(E.SALARIO) < 1500;
```

2.1.39 ROLLUP

Usa-se o ROLLUP para produzir linhas superagregadas pela referência cruzada de colunas. O agrupamento ROLLUP produz um conjunto de resultados que contém as linhas agrupadas normais e os valores de subtotais.

2.1.39.1 Práticas

Exibir o somatório de Salário agrupado por Departamento e Cargo dos empregados com Número de Departamento menor que 60, exibindo os subtotais de Departamento e Cargo.

--Todos

```
SELECT CAST(DEPARTAMENTO AS CHAR), CAST(CARGO AS CHAR),
SUM(SALARIO)
FROM EMPREGADO
WHERE DEPARTAMENTO < 60
GROUP BY DEPARTAMENTO, CARGO
UNION
SELECT '', 'TOTAL', COALESCE(SUM(SALARIO), 0)
FROM EMPREGADO;
```

--SQLServer | Oracle

```
SELECT DEPARTAMENTO, CARGO, SUM(SALARIO)
FROM EMPREGADO
WHERE DEPARTAMENTO < 60
GROUP BY ROLLUP (DEPARTAMENTO, CARGO);
```

--MySQL

```
SELECT DEPARTAMENTO, CARGO, SUM(SALARIO)
FROM EMPREGADO
WHERE DEPARTAMENTO < 60
GROUP BY DEPARTAMENTO, CARGO WITH ROLLUP;
```

2.1.40 CUBE

Usa-se o CUBE para produzir um conjunto de resultados que contém as linhas de ROLLUP e as linhas de tabulação cruzada. Pode-se usá-lo para produzir valores de tabulação cruzada com uma instrução SELECT simples.

2.1.40.1 Práticas

Exibir o somatório de Salário agrupado por Departamento e Cargo dos empregados com Número de Departamento menor que 60, exibindo a referência cruzada de Departamento e Cargo.

```
--SQLServer | Oracle  
SELECT DEPARTAMENTO, CARGO, SUM(SALARIO)  
FROM EMPREGADO  
WHERE DEPARTAMENTO < 60  
GROUP BY CUBE (DEPARTAMENTO, CARGO);
```

2.1.41 Ordenação de Dados

Usa-se o ORDER BY para classificar as linhas de uma seleção.

2.1.41.1 Práticas

Exibir a média de salários pagos por Departamento em ordem decrescente.

```
SELECT D.NOME, AVG(E.SALARIO) AS MEDIAS  
FROM DEPARTAMENTO D INNER JOIN EMPREGADO E  
ON D.NUMERO = E.DEPARTAMENTO  
GROUP BY D.NOME  
ORDER BY AVG(E.SALARIO) DESC;
```

Exibir o somatório dos salários pagos por Cargo em ordem ascendente.

```
SELECT C.NOME, SUM(E.SALARIO) AS MEDIAS  
FROM CARGO C INNER JOIN EMPREGADO E  
ON C.NUMERO = E.CARGO  
GROUP BY C.NOME  
ORDER BY 1 ASC;
```

```
SELECT D.NOME, AVG(E.SALARIO) AS MEDIAS  
FROM DEPARTAMENTO D INNER JOIN EMPREGADO E  
ON D.NUMERO = E.DEPARTAMENTO  
GROUP BY D.NOME  
ORDER BY AVG(E.SALARIO) DESC;
```

```
SELECT C.NOME, SUM(E.SALARIO) AS MEDIAS
```

```
FROM CARGO C INNER JOIN EMPREGADO E  
ON C.NUMERO = E.CARGO  
GROUP BY C.NOME  
ORDER BY 1 ASC;
```

2.1.42 Aninhamento de Grupo de Dados

As funções de grupo podem ser aninhadas até uma profundidade de dois.

2.1.42.1 Práticas

Exibir o salário médio máximo pago por Departamento.

```
--Oracle  
SELECT MAX(AVG(E.SALARIO)) AS MEDIAS  
FROM EMPREGADO E  
GROUP BY E.DEPARTAMENTO;
```

2.1.43 Subconsulta

Consiste em executar uma consulta (externa) a partir de outra consulta (interna).

2.1.43.1 Dicas

Existem operadores que retornam apenas uma linha (=, <>, >, >=, <, <=) e o que retorna várias linhas (IN);

Coloque subconsultas entre parênteses;

A quantidade de níveis de subconsultas dependem do banco de dados.

2.1.43.2 Práticas

Quais os empregados possuem salário maior que o de Pedroca?

```
SELECT APELIDO, SALARIO  
FROM EMPREGADO  
WHERE SALARIO >  
(SELECT SALARIO FROM EMPREGADO WHERE APELIDO='PEDROCA');
```

Qual o nome (Primeironome + Sobrenome) dos empregados possuem o menor salário?

```
--SQLSever  
SELECT PRIMEIRONOME + ' ' + SOBRENOME AS NOME, SALARIO FROM  
EMPREGADO  
WHERE SALARIO = (SELECT MIN(SALARIO) FROM EMPREGADO);
```

```
--Postgre | Oracle | Firebird  
SELECT PRIMEIRONOME || ' ' || SOBRENOME AS NOME, SALARIO FROM  
EMPREGADO  
WHERE SALARIO = (SELECT MIN(SALARIO) FROM EMPREGADO);
```

--MySQL

```
SELECT CONCAT(PRIMEIRONOME, ' ', SOBRENOME) AS NOME, SALARIO  
FROM EMPREGADO  
WHERE SALARIO = (SELECT MIN(SALARIO) FROM EMPREGADO);
```

Exibir o menor Salário por Departamento onde o menor Salário seja o maior que o menor Salário do Departamento de código 50.

```
SELECT D.NOME, MIN(E.SALARIO)  
FROM EMPREGADO E  
INNER JOIN DEPARTAMENTO D  
ON E.DEPARTAMENTO = D.NUMERO  
GROUP BY D.NOME  
HAVING MIN(E.SALARIO) >  
(SELECT MIN(SALARIO) FROM EMPREGADO WHERE DEPARTAMENTO =  
50);
```

O que há de errado com a instrução abaixo? A subconsulta retornou mais de 1 valor. Isso não é permitido quando a subconsulta segue um =, !=, <, <=, >, >= ou quando ela é usada como uma expressão.

```
SELECT Numero, Sobrenome  
FROM Empregado  
WHERE Salario = (SELECT MIN(Salario)  
FROM Empregado  
GROUP BY Departamento);
```

2.1.44 ANY

Seleciona algum conjunto de dados.

2.1.44.1 Práticas

Exibir Número, Sobrenome, Salário dos empregados que possuírem algum salário menor que o menor salário do departamento de código 50.

```
SELECT NUMERO, SOBRENOME, SALARIO  
FROM EMPREGADO  
WHERE SALARIO < ANY  
(SELECT MIN(SALARIO) FROM EMPREGADO WHERE DEPARTAMENTO =  
50);
```

2.1.45 ALL

Seleciona todos os dados.

2.1.45.1 Práticas

Exibir Número, Sobrenome, Salário dos todos os empregados que possuírem salário menor que o menor salário do departamento de código 50.

```
SELECT NUMERO, SOBRENOME, SALARIO
FROM EMPREGADO
WHERE SALARIO < ALL
(SELECT MIN(SALARIO) FROM EMPREGADO WHERE DEPARTAMENTO =
50);
```

2.1.46 Correlacionadas

Ocorre quando uma subconsulta faz referência a uma coluna de uma tabela referenciada na instrução mãe. Uma subconsulta correlacionada é avaliada uma vez para cada linha processada pela instrução mãe. A instrução mãe pode ser uma instrução SELECT, UPDATE ou DELETE.

2.1.46.1 Práticas

Exibir todos os empregados (Sobrenome, Salário e Número do Departamento) que ganham acima do salário médio de seus respectivos departamentos.

```
SELECT EXT.SOBRENOME, EXT.SALARIO, EXT.DEPARTAMENTO
FROM EMPREGADO EXT
WHERE EXT.SALARIO >
(SELECT AVG(ITN.SALARIO)
FROM EMPREGADO ITN
WHERE ITN.DEPARTAMENTO = EXT.DEPARTAMENTO);
```

2.1.47 EXISTS

O operador EXISTS testa a existência de linhas no conjunto de resultados da subconsulta. Se o valor da linha de uma subconsulta for encontrado: A pesquisa na consulta interna não continuará; e A condição será indicada por um flag TRUE. Se o valor de linha de uma subconsulta não for encontrado: A condição será indicada por um flag FALSE; A pesquisa continuará na consulta interna.

Práticas

Exibir os empregados (Sobrenome, Salário e Número do Departamento) com no mínimo uma pessoa subordinada a eles.

```
SELECT EXT.SOBRENOME, EXT.SALARIO, EXT.DEPARTAMENTO
FROM EMPREGADO EXT
WHERE EXISTS
(SELECT 'X'
FROM EMPREGADO ITN
WHERE ITN.GERENTE = EXT.NUMERO);
```

2.1.48 União

Retorna todas as linhas selecionadas por uma consulta. Use esse operador para retornar todas as linhas de várias tabelas e eliminar as linhas duplicadas.

2.1.48.1 Dicas

Necessariamente os campos das tabelas unidas devem possuir os mesmos tipos de dados;

Os campos devem estar posicionados na mesma ordem;

A quantidade de campos deve ser a mesma;

A cláusula ORDER BY deve ser utilizada no final da união.

2.1.48.2 Práticas

Exibir a união das tabelas Cargo e Departamento para os campos Número e Nome.

```
SELECT NUMERO, NOME  
FROM CARGO  
UNION  
SELECT NUMERO, NOME  
FROM DEPARTAMENTO;
```

2.1.49 Intersecção

Serve para retornar todas as linhas comuns a várias consultas.

2.1.49.1 Práticas

Exibir a intersecção das tabelas Cargo e Departamento para os campos Número e Nome.

--SQLServer | Postgre | Oracle

```
SELECT NUMERO, NOME FROM CARGO INTERSECT SELECT NUMERO,  
NOME FROM DEPARTAMENTO;
```

--MySQL

```
SELECT NUMERO, NOME FROM CARGO INNER JOIN DEPARTAMENTO  
USING (NUMERO, NOME);
```

2.1.50 Subtração

Serve para obter linhas retornadas pela primeira consulta e ausentes na segunda consulta.

2.1.50.1 Práticas

Exibir a subtração das tabelas Cargo e Departamento para os campos Número e Nome.

--SQLServer | Oracle

```
SELECT NUMERO, NOME FROM CARGO MINUS SELECT NUMERO, NOME  
FROM DEPARTAMENTO;
```

--Postgre

```
SELECT NUMERO, NOME FROM CARGO EXCEPT SELECT NUMERO,  
NOME FROM DEPARTAMENTO;
```

--MySQL

```
SELECT DISTINCT NUMERO, NOME FROM CARGO WHERE (NUMERO,  
NOME) NOT IN (SELECT NUMERO, NOME FROM DEPARTAMENTO);
```

2.1.51 Macrossubstituição

Alguns bancos permitem passagem de parâmetros com o uso de variáveis de substituição.

2.1.51.1 Dicas

No Oracle a macrossubstituição é com (&);

No SQLServer a macrossubstituição é com (:);

2.1.51.2 Práticas

Exibir o Sobrenome do empregado que possuir o Número informado pelo usuário.

```
SELECT SOBRENOME  
FROM EMPREGADO  
WHERE NUMERO = :Informe_Numero;
```

2.1.52 Seleção de Seleção

Alguns BD permitem a seleção de dados nas cláusulas SELECT e/ou FROM.

--Oracle

```
SELECT CARGO FROM  
(SELECT C.NOME AS CARGO, AVG(E.SALARIO) AS MEDIASALARIO  
FROM CARGO C INNER JOIN EMPREGADO E  
ON C.NUMERO = E.CARGO  
GROUP BY C.NOME);
```

--SQLServer

```
SELECT E.NUMERO, E.SOBRENOME, E.SALARIO, E.DEPARTAMENTO AS  
DEPCOD, (SELECT D.NOME FROM DEPARTAMENTO D WHERE D.NUMERO  
= E.DEPARTAMENTO) AS DEPNAME  
FROM EMPREGADO E;
```

2.1.53 Análise Top-N

Alguns BD permitem a seleção das N primeiras linhas que satisfazem a uma determinada seleção.

2.1.53.1 Práticas

Exibir os 2 primeiros registros de empregados cujos os salários sejam menores que 2000.

--SQLServer

```
SELECT TOP 2 NUMERO, APELIDO, SALARIO FROM EMPREGADO  
WHERE SALARIO < 2000;
```

--Postgre | MySQL

```
SELECT NUMERO, APELIDO, SALARIO FROM EMPREGADO  
WHERE SALARIO < 2000 LIMIT 2;
```

--Oracle

```
SELECT NUMERO, APELIDO, SALARIO FROM EMPREGADO  
WHERE SALARIO < 2000 AND ROWNUM <= 2;
```

--Firebird

```
SELECT FIRST 2 NUMERO, APELIDO, SALARIO FROM EMPREGADO  
WHERE SALARIO < 2000;
```

2.1.54 Excluir Registros

Excluir registros.

2.1.54.1 Sintaxe

```
DELETE FROM nome  
    [WHERE definição]  
    [ORDER BY ...];
```

2.1.55 Excluir e recriar tabela:

Apaga e recria uma tabela.

2.1.55.1 Sintaxe

```
TRUNCATE TABLE nome_tabela;
```

2.1.56 Excluir Tabela

Apaga uma tabela.

2.1.56.1 Sintaxe

```
DROP TABLE nome_tabela;
```

2.1.57 Transações

Tratamento de dados de forma atômica.

2.1.57.1 Dicas

Em alguns BD as instruções DDL e DCL são automáticas.

2.1.57.2

2.1.57.3 Práticas

Exibir os dados da tabela Empregado;

Salvar um ponto A;

Apagar todos os empregados do Departamento 50;

Exibir os dados da tabela Empregado;

Salvar um ponto B;

Apagar todos os empregados com Salário = 1000;

Exibir os dados da tabela Empregado;

Restaurar o ponto A;

Exibir os dados da tabela Empregado;

Restaurar o ponto B;

Exibir os dados da tabela Empregado.

--SQLServer

SAVE TRANSACTION A;

SELECT * FROM DEPARTAMENTO;

DELETE FROM DEPARTAMENTO WHERE NUMERO = 110;

SELECT * FROM DEPARTAMENTO;

UPDATE DEPARTAMENTO SET NOME = 'TESTE' WHERE NUMERO = 90;

SELECT * FROM DEPARTAMENTO;

ROLLBACK TRANSACTION A;

SAVE TRANSACTION B;

INSERT INTO DEPARTAMENTO (NUMERO, NOME) VALUES (200, 'TESTE2');

SELECT * FROM DEPARTAMENTO;

COMMIT TRANSACTION B;

SELECT * FROM DEPARTAMENTO;

2.1.58 Visão

Serve para apresentar combinações ou subconjuntos lógicos de dados através de views de tabelas. A view é uma tabela lógica baseada em uma tabela ou em outra view. Ela não contém dados próprios, mas é uma janela por meio da qual os dados das tabelas podem ser vistos e algumas vezes alterados. As tabelas nas quais uma view se baseia são denominadas de tabelas-base. A view é armazenada como uma instrução SELECT no dicionário de dados.

2.1.58.1 Sintaxe

```
CREATE VIEW [ schema_name . ] view_name  
[ (column [ ,...n ] ) ]
```

AS select_statement;

2.1.58.2 Dicas

Serve para restringir o acesso a dados;

Serve para facilitar as consultas complexas;

Serve para permitir a independência dos dados;

Serve para apresentar diferentes visões dos mesmos dados.

2.1.58.3 View Simple

Cria dados a partir de uma tabela apenas;

Não contém funções e nem grupo de dados;

Permite operações DML.

2.1.58.4 Práticas

Criar uma visão denominada VISAO_1, que contenha os campos Número, Primeironome, Sobrenome dos empregados do departamento 80. Depois de criada insira um novo registro nessa visão. Finalizando exiba todos os dados da tabela Empregado.

```
CREATE VIEW VISAO_1 AS  
SELECT NUMERO, PRIMEIRONOME, SOBRENOME FROM EMPREGADO  
WHERE DEPARTAMENTO=80;
```

```
INSERT INTO VISAO_1 VALUES (200, 'ANTONIETA', 'LARANJEIRA');
```

```
SELECT * FROM EMPREGADO;
```

2.1.58.5 View Complexas

Cria dados a partir de várias tabelas;

Contém funções ou grupo de dados;

Nem sempre permite operações DML.

2.1.58.6 Práticas

Criar uma visão denominada VISAO_2, que contenha os campos Nome (Primeironome + ' ' + Sobrenome) e o campo Salariobase (Salário – 500) dos empregados do departamento 30. Depois de criada tente inserir um novo registro nessa visão. Finalizando exiba todos os dados da tabela Empregado.

```
CREATE VIEW VISAO_2 AS  
SELECT (PRIMEIRONOME + ' ' + SOBRENOME) AS NOME, (SALARIO - 500)  
AS SALARIOBASE  
FROM EMPREGADO WHERE DEPARTAMENTO=10;
```

```
INSERT INTO VISAO_2 VALUES ('JULIETA PEREIRA', 400);
```

```
SELECT * FROM EMPREGADO;
```

2.1.59 Controle de Usuários

Criar novos usuários para o BD.

2.1.59.1 Sintaxe

```
CREATE LOGIN login  
WITH PASSWORD = 'senha';  
CREATE USER usuario  
FOR LOGIN [operador]  
WITH DEFAULT_SCHEMA = [dbo];  
Oracle  
CREATE USER usuario  
IDENTIFIED BY 'senha';
```

2.1.59.2 Práticas

Para adicionar um usuário ao banco de dados do SQL Server você tem que seguir três passos: Primeiro: você deve criar um login, que é um "cara" que tem permissões de se logar no SQL Sever

```
CREATE LOGIN USUARIO WITH PASSWORD = 'senha';
```

Segundo: você deve criar um usuário para o banco de dados que deseja mapeando esse usuário para o login criado, assim seu usuário conseguirá se logar no SQL Server e entrar no banco de dados desejado.

```
CREATE USER USUARIO FROM LOGIN USUARIO;
```

Terceiro: você deve dar ou remover permissões ao usuário porque até o segundo passo o usuário criado só tem direito a entrar no banco de dados, dando as permissões o usuário já pode operar no banco de dados.

Se o usuário for comum você pode adicioná-lo apenas as roles de db_reader e db_writer, que permitirá que o usuário faça select, insert, delete e update em todas as tabelas do referido banco de dados.

```
EXEC SP_ADDROLEMEMBER 'DB_DATAREADER', 'USUARIO'
```

2.1.60 Privilégios

Fornecer acesso aos objetos dos BD.

2.1.60.1 Sintaxe

```
GRANT privilégio ON objeto TO usuário;  
REVOKE privilégio ON objeto FROM usuário;
```

2.1.60.2 Práticas

Fornecer acesso aos objetos dos BD.

```
GRANT CREATE TABLE, CREATE VIEW TO operador1;  
GRANT SELECT ON EMPREGADO TO operador1;
```

Retirar acesso aos objetos dos BD.

```
REVOKE CREATE TABLE, CREATE VIEW FROM operador1;  
REVOKE SELECT ON EMPREGADO FROM operador1;
```

2.1.61 Sequências

Alguns bancos como o Postgre e o Oracle possuem um objeto chamado de sequência que equivale no SQLServer a regra de IDENTITY(1, 1).

2.1.61.1 Sintaxe

```
CREATE [TEMPORARY | TEMP] SEQUENCE name  
[INCREMENT [BY] increment ]  
[MINVALUE minvalue | NO MINVALUE]  
[MAXVALUE maxvalue | NO MAXVALUE]  
[START [ WITH ] start]  
[CACHE cache]  
[[ NO ] CYCLE]
```

2.1.62 Vínculos

A grande vantagem dos vínculos de banco de dados é que eles permitem aos usuários acessarem dados de um banco de dados remoto de modo que eles fiquem conectados por meio do conjunto de privilégios do proprietário do objeto. Em outras palavras, um usuário local pode acessar um banco de dados remoto sem necessariamente ser um usuário do banco de dados remoto.

2.1.62.1 Práticas

Vincular dados.

```
CREATE PUBLIC DATABASE LINK banco  
USING 'usuário';  
SELECT * FROM objeto@banco;
```

2.2 Objetos de Bancos de Dados

Postgre:

- Tables
- Views
- Functions
- Domains

- Rules
- Triggers
- Indices
- Sequences
- Composite Types
- Enum Types
- Base Types
- Aggregates
- Operators

2.3 Principais Funções do SQLServer

ASCII(*String*)

pega o valor em ASCII da *String*

CHAR(integer)

troca inteiro do ASCII em um caractere

LEN(*String*)

Identifica o comprimento de uma expressão em caracteres

LOWER(*String*)

converte uma *String* uppercase para lowercase.

LTRIM(*String*)

remove os espaços em branco

PATINDEX(posicao, expressao)

devolve a posição de uma *String* dentro de um texto. Se não encontrar, retorna zero.

REPLICATE(*String*, integer)

repete N vezes um caractere especificado

REVERSE(*String*)

retorna o inverso de uma expressao

RTRIM(*String*)

remove os espaços em branco à direita de uma *String*

SPACE(integer)

que retorna o número de espaços em branco informados no parâmetro

STUFF(*String* texto, X, Y, *String* texto_a_inserir)

apaga da *String* "texto" os y caracteres a partir da posição x e os substitui por "texto_a_inserir"

SUBSTRING(*String* texto, posicao_inicial, tamanho)
retorna uma *String* com o comprimento definido em "tamanho" extraída da *String* "texto", a partir da "posicao_inicial"

UPPER(*String*)
retorna *String* em maiúsculas

DATEADD (parte, numero, data)
adiciona um valor a parte de uma data

DATEDIFF (parte, data inicial, data final)
subtrai a data inicial da data final, indicando o resultado na unidade definida em "parte"

GETDATE()
retorna a data atual do sistema

DATENAME (parte, data)
retorna o nome da parte de uma data

DATEPART(parte, data)
retorna a parte de uma data

CAST(expressao as datatype)
converte uma expressao no datatype informado

COL_LENGTH(nome_da_tabela, nome_da_coluna)
retorna o tamanho da coluna

COL_NAME(id_da_tabela, id_da_coluna)
retorna o nome da coluna

DATALENGTH(expressao)
retorna o numero de bytes usados para armazenar a expressao

DB_ID(nome_do_banco)
retorna o ID do banco informado

DB_NAME(id_do_banco)
retorna o nome do banco

HOST_ID()

retorna a ID da estação que está acessando o SQL Server

HOST_NAME()

retorna o nome da estação que está acessando o SQL Server

IDENT_INCR(nome_da_tabela_ou_view)

retorna o valor incrementado

IDENT_SEED(tabela_ou_view)

retorna o valor inicial da coluna

INDEX_COL(nome_da_tabela, indice_id, chave_id)

retorna o nome da coluna que participa do índice

ISNULL(expressao, valor)

se a expressao for null, troca pelo valor especificado

ISNUMERIC(expressao)

retorna 1 se a expressao for numerica e 0 se não for

NEWID()

retorna um novo valor do tipo uniqueidentifier

NULLIF(expressao_1, expressao_2)

retorna nulo se as duas expressões forem equivalentes. Se não forem, retorna a primeira expressao.

OBJECT_ID(nome_do_objeto)

retorna o ID de um objeto, a partir do nome fornecido

OBJECT_NAME(ID_do_objeto)

retorna o nome do objeto, a partir do ID fornecido

PARSENAME(objeto, parte)

retorna a parte do nome de um objeto, desde que tenha sido qualificado

STATS_DATE(tabela_id, indice_id)

retorna a data em que as estatísticas do índice foram atualizadas

SUSER_SID(nome_do_usuario)

retorna o ID do usuario informado

SUSER_NAME(usuario_id)

retorna o id do usuário no servidor. O argumento é opcional.

SUSER_SNAME(id_do_usuario)

retorna o nome do usuario informado. Se nenhum ID de usuario for passado para a função, retorna o nome do usuario logado

USER_ID(nome_do_usuario)

retorna o ID do usuario informado para o BD em uso

USER_NAME(id_do_usuario)

retorna o usuario conectado ao BD

ABS(numero)

retorna o valor absoluto do numero

ACOS(float)

retorna o arco-coseno do numero informado

ASIN(float)

retorna o arco-seno do numero informado

ATAN(float)

retorna o arco-tangente do numero informado

ATN2(Float expressao_1, float expressao_2)

arco-tangente do valor definido pela divisão da primeira expressão pela segunda

CEILING(numero)

retorna o menor inteiro que seja maior ou igual ao numero informado

COS(float)

retorna o coseno do numero informado

COT(float)

retorna o cotangente do numero informado

DEGREES(numero)

converte radianos para graus

EXP(float)

retorna o exponencial de um numero especificado

FLOOR(numero)

retorna o maior inteiro que seja menor ou igual ao numero informado

LOG(float)

retorna o logaritmo natural do numero informado

LOG10(float)

retorna o logaritmo base 10 do numero informado

PI()

retorna o valor de PI 3.1415926535897931.

POWER(numero, potencia)

retorna o valor elevado à potencia informada

RADIANS(numero)

converte graus para radianos

RAND(expressao)

um número aleatório entre 0 e 1. Expressão é opcional e será usada como semente da cadeia pseudo-aleatória

ROUND(numero, precisao, arredonda_ou_trancar)

arredonda ou trunca o numero fornecido de acordo com a precisao informada. Se o terceiro parametro não for passado para a função, o numero é arredondado. Se quiser que o numero seja truncado, deve-se fornecer o valor 1

SIGN(numero)

retorna sinal positivo, negativo ou zero do numero

SIN(float)

retorna o seno do angulo especificado

SQRT(float)

retorna a raiz quadrada de um numero

TAN(float)

retorna a tangente de um numero informado

SQUARE(float)

retorna o quadrado de um numero

Principais Funções do Oracle

ABS(n)

Devolve o valor absoluto de (n).

CEIL(n)

Obtém o valor inteiro imediatamente superior ou igual a "n".

FLOOR(n)

Devolve o valor inteiro imediatamente inferior ou igual a "n".

MOD (m, n)

Devolve o resto resultante de dividir "m" entre "n".

NVL (valor, expressão)

Substitui um valor nulo por outro valor.

POWER (m, expoente)

Calcula a potência de um número.

ROUND (numero [, m])

Arredonda números com o número de dígitos de precisão indicados.

SIGN (valor)

Indica o signo do "valor".

SQRT(n)

Devolve a raiz quadrada de "n".

TRUNC (numero, [m])

Trunca números para que tenham uma certa quantidade de dígitos de precisão.

VAIRANCE (valor)

Devolve a média de um conjunto de valores.

CHR(n)

Devolve o caractere cujo valor em binário é equivalente a "n".

CONCAT (cad1, cad2)

Devolve "cad1" concatenada com "cad2".

LOWER (cad)

Devolve a cadeia "cad" em minúsculas.

UPPER (cad)

Devolve a cadeia "cad" em maiúsculas.

INITCAP (cad)

Converte a cadeia "cad" a tipo título.

LPAD (cad1, n[,cad2])

Adiciona caracteres à esquerda da cadeia até que tenha uma certa longitude.

RPAD (cad1, n[,cad2])

Adiciona caracteres à direita até que tenha uma certa longitude.

LTRIM (cad [,set])

Suprime um conjunto de caracteres à esquerda da cadeia.

RTRIM (cad [,set])

Suprime um conjunto de caracteres à direita da cadeia.

REPLACE (cad, cadeia_busca [, cadeia_substitucao])

Substitui um caractere ou caracteres de uma cadeia com 0 ou mais caracteres.

SUBSTR (cad, m [,n])

Obtém parte de uma cadeia.

TRANSLATE (cad1, cad2, cad3)

Converte caracteres de uma cadeia em caracteres diferentes, segundo um plano de substituição marcado pelo usuário.

ASCII(cad)

Devolve o valor ASCII da primeira letra da cadeia "cad".

INSTR (cad1, cad2 [, comeco [,m]])

Permite uma busca de um conjunto de caracteres em uma cadeia, mas não suprime nenhum caractere depois.

LENGTH (cad)

Devolve o número de caracteres de cad.

SYSDATE

Devolve a data do sistema.

ADD_MONTHS (data, n)

Devolve a data "data" incrementada em "n" meses.

LASTDAY (data)

Devolve a data do último dia do mês que contém "data".

MONTHS_BETWEEN (data1, data2)

Devolve a diferença em meses entre as datas "data1" e "data2".

NEXT_DAY (data, cad)

Devolve a data do primeiro dia da semana indicado por "cad" depois da data indicada por "data".

TO_CHAR

Transforma um tipo DATE ou NUMBER em uma cadeia de caracteres.

TO_DATE

Transforma um tipo NUMBER ou CHAR em DATE.

TO_NUMBER

Transforma uma cadeia de caracteres em NUMBER.

2.4 Tipos de Dados

SQLServer

- BIGINT: valores numéricos inteiros.
- BIT: bits ou seja somente poderá conter os valores lógicos 0 ou 1.
- CHAR(N): N caracteres fixos no formato não Unicode, com espaços em branco.
- DATE: somente data.
- DATETIME: data e hora, com precisão de centésimos de segundos.
- DECIMAL(18,0): mesmas funcionalidades do tipo NUMERIC.
- FLOAT: valores numéricos aproximados com precisão de ponto flutuante.
- IMAGE: dados no formato binário.
- INT: valores numéricos inteiros.
- NCHAR(N): N caracteres fixos no formato Unicode, com espaços em branco.
- NTEXT: caracteres no formato Unicode.
- NUMERIC(18,0): valores numéricos com casas decimais, utilizando precisão.
- NVARCHAR(N): Armazena N caracteres no formato Unicode.
- REAL: valores numéricos aproximados com precisão de ponto flutuante.
- SMALLDATETIME: data e hora, com precisão de minutos.
- SMALLINT: valores numéricos inteiros.
- TEXT: caracteres no formato não Unicode.
- TIME: somente hora. Pode armazenar segundos até a fração de 9999999
- TINYINT: valores numéricos inteiros.
- VARCHAR(N): N caracteres (até 8.000) no formato não Unicode.

Postgre

- bigint: inteiro de oito bytes com sinal.
- bit: cadeia de bits de comprimento fixo.
- bool: booleano lógico (verdade/falso).
- bytea: dados binários.
- char(n): cadeia de caracteres de comprimento fixo.
- date: data de calendário (ano, mês,dia).

- decimal[(p, s)] : numérico exato com precisão selecionável.
- float8: número de ponto flutuante de precisão dupla.
- Int: inteiro de quatro bytes com sinal.
- numeric[(p, s)]: numérico exato com precisão selecionável.
- real: número de ponto flutuante de precisão simples.
- smallint: inteiro de dois bytes com sinal.
- text: cadeia de caracteres de comprimento variável.
- time: hora do dia.
- timestamp: data e hora.
- varbit(n): cadeia de bits de comprimento variável.
- varchar(n): cadeia de caracteres de comprimento variável.

Oracle

- BLOB, CLOB, NCLOB e BFILE: Tipos de dados para conteúdos binários.
- CHAR(n): Conjunto de caracteres de tamanho fixo.
- DATE: Um valor de data.
- LONG RAW: Dados binários..
- LONG: Conjunto de caracteres de tamanho variável.
- NUMBER(p, e): Representa um número.
- VARCHAR2(n): Conjunto de caracteres de tamanho variável.

MySQL

- BIGINT: inteiros grandes.
- BLOB: BLOB normal.
- BOOL: Booleano.
- CHAR: o mesmo que CHAR(1).
- DATE: data. Exibido como YYYY-MM-DD.
- DATETIME: data e hora. Exibido como YYYY-MM-DD HH:MM:SS.
- DECIMAL[(M,D)]: número de ponto flutuante armazenado como char.
- DOUBLE: números de ponto flutuante de precisão dupla.
- FLOAT(precisão): números de ponto flutuante de precisão simples ou dupla.
- INT: inteiros regulares.
- INTEGER: o mesmo que INT.
- LONGBLOB: BLOB longo.
- LONGTEXT: TEXT longo.
- SMALLINT: inteiros pequenos.
- TEXT: TEXT normal.
- TIME: hora. Exibido como HH:MM:SS.
- TINYBLOB: BLOB pequeno.
- TINYTEXT: TEXT pequeno.
- VARCHAR(n): variável *String* de tamanho variável.

2.5 Usuários e Senhas Padrões

SQLServer: Usuário sa; Senha definida.

Postgre: Usuário root; Sem senha.

Oracle: Usuário SYSTEM; Senha manager.

MySQL: Usuário root; Sem senha.

Firebird: Usuário sysdba; Senha masterkey.

2.6 Exemplos

```
CREATE TABLE DISCIPLINA (COD INTEGER, DSC VARCHAR(50), PRIMARY
KEY(COD));
CREATE TABLE PROFESSOR (MAT INTEGER, NOM VARCHAR(50),
PRIMARY KEY(MAT));
CREATE TABLE ALUNO (MAT INTEGER, NOM VARCHAR(50), PRIMARY
KEY(MAT));
CREATE TABLE NOTA (MAT INTEGER, DIS INTEGER, NT1 FLOAT, NT2
FLOAT, NT3 FLOAT, PRIMARY KEY (MAT,DIS), FOREIGN KEY (MAT)
REFERENCES ALUNO (MAT), FOREIGN KEY (DIS) REFERENCES
DISCIPLINA (COD) ON DELETE CASCADE ON UPDATE CASCADE);
DROP TABLE PROFESSOR;
CREATE TABLE TEMPOR (MED FLOAT);
```

```
CREATE USER 'U1'@'localhost' IDENTIFIED BY 'u1';
CREATE USER 'U2'@'localhost' IDENTIFIED BY 'u2';
CREATE USER 'U3'@'localhost' IDENTIFIED BY 'u3';
GRANT ALL PRIVILEGES ON *.* TO 'U1'@'localhost';
GRANT INSERT ON exemplo.disciplina TO 'U2'@'localhost';
GRANT UPDATE ON exemplo.disciplina TO 'U2'@'localhost';
GRANT SELECT ON exemplo.disciplina TO 'U2'@'localhost';
GRANT DELETE ON exemplo.disciplina TO 'U3'@'localhost';
REVOKE DELETE ON exemplo.disciplina TO 'U3'@'localhost'
FLUSH PRIVILEGES;
DROP USER 'U3'@'localhost';
```

```
START TRANSACTION;
SELECT @A:=AVG(NT1) FROM NOTA WHERE MAT=1;
UPDATE TEMPOR SET MED=@A;
COMMIT;
```

```
CREATE TRIGGER GATILHO BEFORE INSERT ON NOTA
FOR EACH ROW UPDATE NT3 SET NT3 = (NT1 + NT2) / 2;
```

2.7 Exercícios

Obs: a solução foi realizada no SQLServer.

--Tabela Cargos

```
CREATE TABLE CARGO (  
  NUMERO INTEGER IDENTITY( 1, 1),  
  NOME VARCHAR(30) NOT NULL,  
  PRIMARY KEY (NUMERO)  
);  
INSERT INTO CARGO (NOME) VALUES ('ANALISTA TRAINEE');  
INSERT INTO CARGO (NOME) VALUES ('ANALISTA JUNIOR');  
INSERT INTO CARGO (NOME) VALUES ('ANALISTA PLENO');  
INSERT INTO CARGO (NOME) VALUES ('ANALISTA SÊNIOR');  
INSERT INTO CARGO (NOME) VALUES ('ANALISTA MASTER');  
INSERT INTO CARGO (NOME) VALUES ('CONSULTOR');  
SELECT * FROM CARGO;
```

--Tabela Departamentos

```
CREATE TABLE DEPARTAMENTO (  
  NUMERO INTEGER,  
  NOME VARCHAR(30) NOT NULL,  
  FATOR DECIMAL(3,2),  
  PRIMARY KEY (NUMERO)  
);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (10,  
'ADMINISTRAÇÃO (ADMIN)', 1.75);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (20,  
'MARKETING (MKT)', 1.85);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (50,  
'COMPRAS (COM)', 2.73);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (60,  
'TECNOLOGIA (TEC)', 1.55);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (80,  
'VENDAS (VEN)', 1.87);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (90,  
'EXECUTIVO (EXEC)', 1.92);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (110,  
'CONTABILIDADE (CONTAB)', 2.35);  
INSERT INTO DEPARTAMENTO (NUMERO, NOME, FATOR) VALUES (190,  
'CONTRATOS (CTT)', 2.23);  
  
SELECT * FROM DEPARTAMENTO;
```

--Tabela Empregados

```
CREATE TABLE EMPREGADO (  
  NUMERO INTEGER,  
  PRIMEIRONOME VARCHAR(25),  
  SOBRENOME VARCHAR(25),  
  CARGO INTEGER,  
  DEPARTAMENTO INTEGER,  
  SALARIO REAL,  
  COMISSAO INTEGER,  
  NASCIMENTO DATE,  
  APELIDO VARCHAR(10),  
  GERENTE INTEGER,  
  PRIMARY KEY (NUMERO),  
  FOREIGN KEY (CARGO) REFERENCES CARGO (NUMERO) ON UPDATE  
  CASCADE,  
  FOREIGN KEY (DEPARTAMENTO) REFERENCES DEPARTAMENTO  
  (NUMERO) ON UPDATE CASCADE  
);
```

```
INSERT INTO EMPREGADO (NUMERO, PRIMEIRONOME, SOBRENOME)  
VALUES (100, 'ANTÔNIO', 'PALM');  
INSERT INTO EMPREGADO (NUMERO, PRIMEIRONOME, SOBRENOME)  
VALUES (101, 'JOSÉ', 'FIGUEIRA');  
INSERT INTO EMPREGADO (NUMERO, PRIMEIRONOME, SOBRENOME)  
VALUES (102, 'MARIA', 'MANGUEIRA');  
INSERT INTO EMPREGADO (NUMERO, PRIMEIRONOME, SOBRENOME)  
VALUES (103, 'JOÃO', 'ABACATEIRO');  
INSERT INTO EMPREGADO (NUMERO, PRIMEIRONOME, SOBRENOME)  
VALUES (104, 'CARLOS', 'CAST');  
INSERT INTO EMPREGADO (NUMERO, PRIMEIRONOME, SOBRENOME)  
VALUES (107, 'MÔNICA', 'VIDEIRA');  
INSERT INTO EMPREGADO (NUMERO, PRIMEIRONOME, SOBRENOME)  
VALUES (124, 'PEDRO', 'LARANJA');  
INSERT INTO EMPREGADO (NUMERO, PRIMEIRONOME, SOBRENOME)  
VALUES (141, 'TIAGO', 'PESSEGUE');  
INSERT INTO EMPREGADO (NUMERO, PRIMEIRONOME, SOBRENOME)  
VALUES (142, 'RENATA', 'VIDEIRA');  
INSERT INTO EMPREGADO (NUMERO, PRIMEIRONOME, SOBRENOME)  
VALUES (143, 'GABRIEL', 'MACIEIRA');
```

```
UPDATE EMPREGADO SET CARGO=7, DEPARTAMENTO=10,  
  GERENTE=NULL, SALARIO=1000, COMISSAO=NULL,  
  NASCIMENTO='01/07/80', APELIDO='TONHO' WHERE NUMERO=100;  
UPDATE EMPREGADO SET CARGO=7, DEPARTAMENTO=20,  
  GERENTE=100, SALARIO=1500, COMISSAO=NULL,  
  NASCIMENTO='18/05/70', APELIDO='ZÉ' WHERE NUMERO=101;
```

```
UPDATE EMPREGADO SET CARGO=8, DEPARTAMENTO=10,  
GERENTE=100, SALARIO=2000, COMISSAO=3, NASCIMENTO='22/07/80',  
APELIDO='MA' WHERE NUMERO=102;  
UPDATE EMPREGADO SET CARGO=9, DEPARTAMENTO=50,  
GERENTE=100, SALARIO=1500, COMISSAO=2, NASCIMENTO='03/03/71',  
APELIDO='JO' WHERE NUMERO=103;  
UPDATE EMPREGADO SET CARGO=NULL, DEPARTAMENTO=60,  
GERENTE=100, SALARIO=2500, COMISSAO=3, NASCIMENTO='01/01/65',  
APELIDO='CA' WHERE NUMERO=104;  
UPDATE EMPREGADO SET CARGO=7, DEPARTAMENTO=90,  
GERENTE=NULL, SALARIO=1000, COMISSAO=2, NASCIMENTO='15/02/60',  
APELIDO='MO' WHERE NUMERO=107;  
UPDATE EMPREGADO SET CARGO=8, DEPARTAMENTO=80,  
GERENTE=107, SALARIO=1100, COMISSAO=NULL,  
NASCIMENTO='30/07/83', APELIDO='PEDROCA' WHERE NUMERO=124;  
UPDATE EMPREGADO SET CARGO=9, DEPARTAMENTO=110,  
GERENTE=107, SALARIO=2000, COMISSAO=2, NASCIMENTO='21/06/78',  
APELIDO='TI' WHERE NUMERO=141;  
UPDATE EMPREGADO SET CARGO=10, DEPARTAMENTO=80,  
GERENTE=107, SALARIO=1500, COMISSAO=NULL,  
NASCIMENTO='02/12/72', APELIDO='RE' WHERE NUMERO=142;  
UPDATE EMPREGADO SET CARGO=NULL, GERENTE=110, SALARIO=2000,  
COMISSAO=2, NASCIMENTO='11/11/81', APELIDO='BIEL' WHERE  
NUMERO=143;  
SELECT * FROM EMPREGADO;
```

--Questões:

--1.Mostre todos os dados dos empregados que nasceram após o ano de 1980.

```
SELECT * FROM EMPREGADO WHERE YEAR(NASCIMENTO) > 1980;
```

--2.Mostre o sobrenome, o cargo, o salário e a comissão dos empregados que recebem comissão.

```
SELECT SOBRENOME, CARGO, SALARIO, COMISSAO FROM EMPREGADO  
WHERE COMISSAO IS NOT NULL;
```

--3.Mostre os empregados que não tem comissão com um aumento de 10% em seus salários, arredondando-os.

```
SELECT NUMERO, APELIDO, COMISSAO, SALARIO, SALARIO + (SALARIO  
*.10) AS BRUTO FROM EMPREGADO WHERE COMISSAO IS NULL;
```

--4.Mostre os sobrenomes de todos os empregados juntamente com o número de anos e de meses completados de nascimento.

```
SELECT SOBRENOME, NASCIMENTO, GETDATE() AS DATACORRENTE,  
DATEDIFF(YEAR,NASCIMENTO,GETDATE()) AS ANOS,  
DATEDIFF(MONTH,NASCIMENTO,GETDATE()) AS MESES FROM  
EMPREGADO
```

--5. Mostre os empregados cujos os sobrenomes iniciem com J, K, L ou M.

```
SELECT SOBRENOME FROM EMPREGADO WHERE SOBRENOME LIKE 'J%'  
OR SOBRENOME LIKE 'K%' OR SOBRENOME LIKE 'L%' OR SOBRENOME  
LIKE 'M%';
```

--6. Mostre todos os funcionários e indique, através de “Sim” ou “Não”, se eles recebem ou não uma comissão.

```
SELECT APELIDO, CASE WHEN COMISSAO IS NULL THEN 'NÃO' ELSE 'SIM'  
END AS COMISSAO FROM EMPREGADO;
```

--7. Mostre os nomes de departamentos, os nomes dos empregados, os nomes dos cargos e os salários dos empregados que trabalham nos departamentos 50, 60 ou 90.

```
SELECT E.PRIMEIRONOME AS EMPREGADO, D.NOME AS  
DEPARTAMENTO, C.NOME AS CARGO, E.SALARIO  
FROM EMPREGADO E  
INNER JOIN DEPARTAMENTO D ON E.DEPARTAMENTO = D.NUMERO  
INNER JOIN CARGO C ON E.CARGO = C.NUMERO  
WHERE E.DEPARTAMENTO IN (50, 60, 90);
```

--8. Quantos empregados têm um primeironome terminado em “o” e em “a”?

```
SELECT 'O' AS TERMINA, COUNT(NUMERO) AS QTDE FROM EMPREGADO  
WHERE PRIMEIRONOME LIKE '%O'  
UNION  
SELECT 'A' AS TERMINA, COUNT(NUMERO) AS QTDE FROM EMPREGADO  
WHERE PRIMEIRONOME LIKE '%A';
```

--9. Mostre o nome de todos os departamentos, além dos números dos empregados que trabalham em cada departamento. Certifique-se de que os departamentos sem empregados também sejam incluídos.

```
SELECT D.NUMERO,  
D.NOME,  
(SELECT COUNT(E.NUMERO) FROM EMPREGADO E WHERE  
E.DEPARTAMENTO = D.NUMERO) AS QTDE_EMP  
FROM DEPARTAMENTO D;
```

--10. Quais cargos são encontrados nos departamentos 10 e 50?

```
SELECT DISTINCT C.NOME FROM EMPREGADO E INNER JOIN CARGO C  
ON E.CARGO = C.NUMERO WHERE E.DEPARTAMENTO IN (10,50);
```

--11.Quais cargos são encontrados nos departamentos administrativo e executivo, e quantos empregados desempenham esses cargos? Mostre primeiro o cargo com o maior número de empregados.

```
SELECT E.DEPARTAMENTO, C.NOME,  
(SELECT COUNT(E.NUMERO) FROM EMPREGADO E WHERE E.CARGO =  
C.NUMERO) AS QTDE_EMP  
FROM EMPREGADO E  
INNER JOIN CARGO C ON E.CARGO = C.NUMERO  
WHERE E.DEPARTAMENTO IN (SELECT X.NUMERO FROM  
DEPARTAMENTO X WHERE X.NOME LIKE 'ADM%' OR X.NOME LIKE  
'EXE%');
```

--12.Mostre todos os empregados que nasceram na primeira metade do mês (antes do dia 16).

```
SELECT E.APELIDO, E.NASCIMENTO FROM EMPREGADO E WHERE  
DAY(E.NASCIMENTO) < 16;
```

--13.Mostre o primeironome, o salário e o número de todos os empregados.

```
SELECT E.PRIMEIRONOME, E.SALARIO, E.NUMERO FROM EMPREGADO  
E;
```

--14.Mostre todos os empregados que têm gerentes com um salário acima ou igual a 1000. Mostre os seguintes dados: primeironome do empregado e do gerente e salário do gerente.

```
SELECT E.APELIDO, E.SALARIO, E.GERENTE FROM EMPREGADO E  
WHERE E.GERENTE IS NOT NULL  
AND EXISTS (SELECT 'X' FROM EMPREGADO E WHERE  
E.GERENTE IS NULL AND E.SALARIO >= 1000);
```

--15.Mostre o número do departamento, o nome, o número do empregado e o salário médio de todos os departamentos juntamente com os nomes, os salários e os cargos dos empregados que trabalham em cada departamento.

```
SELECT      E.NUMERO,  
            E.APELIDO,  
            E.SALARIO,  
            C.NOME AS CARGO,  
            E.DEPARTAMENTO,  
            D.NOME,  
            (SELECT AVG(X.SALARIO) FROM EMPREGADO X WHERE  
X.DEPARTAMENTO = E.DEPARTAMENTO) AS SAL_MED  
FROM EMPREGADO E  
INNER JOIN DEPARTAMENTO D ON E.DEPARTAMENTO = D.NUMERO  
INNER JOIN CARGO C ON E.CARGO = C.NUMERO;
```

--16.Mostre o número do departamento e o menor salário e o maior salário.

```
SELECT      D.NUMERO,  
            (SELECT MIN(E.SALARIO) FROM EMPREGADO E WHERE  
E.DEPARTAMENTO = D.NUMERO) AS MENOR,  
            (SELECT MAX(E.SALARIO) FROM EMPREGADO E WHERE  
E.DEPARTAMENTO = D.NUMERO) AS MAIOR  
FROM DEPARTAMENTO D;
```

--17. Mostre os números, os nomes dos departamentos em que não há empregados trabalhando.

```
SELECT D.NUMERO, D.NOME FROM DEPARTAMENTO D WHERE  
D.NUMERO NOT IN (SELECT E.DEPARTAMENTO FROM EMPREGADO E);
```

--18. Mostre os números, os nomes dos departamentos e os números dos empregados trabalhando em cada departamento que inclua mais de 2 empregados, inclusive.

```
SELECT      D.NUMERO,  
            D.NOME,  
            (SELECT COUNT(E.NUMERO) FROM EMPREGADO E  
WHERE E.DEPARTAMENTO = D.NUMERO) AS QTDE_EMP  
FROM DEPARTAMENTO D  
WHERE (SELECT COUNT(E.NUMERO) FROM EMPREGADO E WHERE  
E.DEPARTAMENTO = D.NUMERO) >= 2;
```

--19. Mostre os números, os nomes dos departamentos e os números dos empregados trabalhando em cada departamento que tenha o maior número de empregados.

```
SELECT TOP 1  
      D.NUMERO,  
      D.NOME,  
      (SELECT COUNT(E.NUMERO) FROM EMPREGADO E WHERE  
E.DEPARTAMENTO = D.NUMERO) AS QTDE_EMP  
FROM DEPARTAMENTO D  
ORDER BY 3 DESC;
```

--20. Mostre os números, os nomes dos departamentos e os números dos empregados trabalhando em cada departamento que tenha o menor número de empregados.

```
SELECT TOP 1  
      D.NUMERO,  
      D.NOME,  
      (SELECT COUNT(E.NUMERO) FROM EMPREGADO E WHERE  
E.DEPARTAMENTO = D.NUMERO) AS QTDE_EMP  
FROM DEPARTAMENTO D  
ORDER BY 3;
```

--21. Mostre o número do funcionário, o sobrenome, o salário, o número do departamento e o salário médio de seu departamento para todos os funcionários.

```
SELECT      E.NUMERO,
            E.SOBRENOME,
            E.SALARIO,
            D.NOME AS DEPARTAMENTO,
            (SELECT AVG(X.SALARIO) FROM EMPREGADO X WHERE
X.DEPARTAMENTO = E.DEPARTAMENTO) AS SAL_MED
FROM EMPREGADO E
INNER JOIN DEPARTAMENTO D ON E.DEPARTAMENTO = D.NUMERO;
```

--22. Mostre todos os funcionários nascidos no dia da semana em que foi correspondente ao número de empregados.

```
SELECT E.APELIDO, DAY(E.NASCIMENTO) AS DIA FROM EMPREGADO E
WHERE DAY(E.NASCIMENTO) = (SELECT COUNT(X.NUMERO) FROM
EMPREGADO X);
```

--23. Crie uma visão geral de aniversários com base na data de nascimento dos empregados. Classifique os aniversários em ordem crescente.

```
SELECT E.NUMERO, E.APELIDO, E.NASCIMENTO FROM EMPREGADO E
ORDER BY E.NASCIMENTO ASC;
```

--24. Crie uma consulta composta para produzir uma lista de empregados que mostre percentuais de aumento, número dos empregados.

--Considerando que os empregados dos departamentos 10, 50 e 110 receberão aumento de 5%,

--os do departamento 60 receberão um aumento de 10%,

--os dos departamento 20 e 80 receberão um aumento de 15%

--e os do departamento 90 não receberão aumento.

```
SELECT      E.NUMERO,
            E.APELIDO,
            E.SALARIO,
            E.DEPARTAMENTO,
            CASE
                WHEN E.DEPARTAMENTO IN (10,50,110) THEN E.SALARIO +
(E.SALARIO * .5)
                WHEN E.DEPARTAMENTO IN (60) THEN E.SALARIO + (E.SALARIO *
.10)
                WHEN E.DEPARTAMENTO IN (20,80) THEN E.SALARIO +
(E.SALARIO * .15)
                WHEN E.DEPARTAMENTO IN (90) THEN E.SALARIO
            END AS AUMENTO
FROM EMPREGADO E;
```

--25.Exiba a data corrente.

```
SELECT TOP 1 GETDATE() FROM CARGO;
```

--26.Exiba o tempo em anos, mês e dias decorridos do nascimento até a data corrente.

```
SELECT  
    E.APELIDO,  
    E.NASCIMENTO,  
    DATEDIFF(YEAR, E.NASCIMENTO, GETDATE()) AS ANOS,  
    DATEDIFF(MONTH, E.NASCIMENTO, GETDATE()) AS MESES,  
    DATEDIFF(DAY, E.NASCIMENTO, GETDATE()) AS DIAS  
FROM EMPREGADO E
```

--27.Crie uma consulta para exibir os seguintes itens referentes aos departamentos:

--o número seja maior que 80 com o salário total de cada cargo em um departamento.

```
SELECT D.NUMERO, D.NOME AS DEPARTAMENTO, C.NOME AS CARGO,  
SUM(E.SALARIO) AS SAL_TOT  
FROM EMPREGADO E  
INNER JOIN DEPARTAMENTO D  
ON E.DEPARTAMENTO = D.NUMERO  
INNER JOIN CARGO C  
ON E.CARGO = C.NUMERO  
WHERE D.NUMERO > 80  
GROUP BY D.NUMERO, D.NOME, C.NOME;
```

--28.Crie uma consulta para exibir os seguintes itens referentes aos departamentos:

--o número seja maior que 80 com o salário total.

```
SELECT D.NUMERO, D.NOME AS DEPARTAMENTO, SUM(E.SALARIO) AS  
SAL_TOT  
FROM EMPREGADO E  
INNER JOIN DEPARTAMENTO D  
ON E.DEPARTAMENTO = D.NUMERO  
INNER JOIN CARGO C  
ON E.CARGO = C.NUMERO  
WHERE D.NUMERO > 80  
GROUP BY D.NUMERO, D.NOME;
```

--29.Crie uma consulta para exibir os seguintes itens referentes aos departamentos:

--o número seja maior que 80 com o salário total de cada cargo, independente do departamento.

```
SELECT SUM(E.SALARIO) AS SAL_TOT
```

```
FROM EMPREGADO E
INNER JOIN DEPARTAMENTO D
ON E.DEPARTAMENTO = D.NUMERO
INNER JOIN CARGO C
ON E.CARGO = C.NUMERO
WHERE D.NUMERO > 80;
```

--30.Crie uma consulta que exiba o agrupamento por departamento e por cargo.

```
SELECT E.DEPARTAMENTO, E.CARGO
FROM EMPREGADO E
GROUP BY E.DEPARTAMENTO, E.CARGO;
```

--31.Crie uma consulta para exibir os 2 empregados que ganham mais.

```
SELECT TOP 2 E.APELIDO, E.SALARIO
FROM EMPREGADO E
ORDER BY E.SALARIO DESC;
```

--32.Crie uma consulta para exibir o número, o sobrenome, o nascimento e o salário dos empregados com os nomes de seus respectivos gerentes.

```
SELECT E.NUMERO, E.SOBRENOME AS EMPREGADO, E.NASCIMENTO,
E.SALARIO, G.SOBRENOME AS GERENTE
FROM EMPREGADO E
LEFT JOIN EMPREGADO G ON E.GERENTE = G.NUMERO;
```

--33.Crie uma consulta hierárquica para exibir o número do empregado, o número do gerente e o sobrenome de todos os empregados que estejam um nível abaixo do gerente.

```
SELECT E.NUMERO, 'GERENTE: ' + E.SOBRENOME AS RECURSO,
E.NASCIMENTO, E.SALARIO
FROM EMPREGADO E
WHERE E.GERENTE IS NULL
UNION
SELECT E.NUMERO, 'EMPREGADO: ' + E.SOBRENOME AS RECURSO,
E.NASCIMENTO, E.SALARIO
FROM EMPREGADO E
WHERE E.GERENTE IS NOT NULL;
```

--34.Exibir os 2 primeiros registros de empregados cujos os salários sejam maiores que 1500.

```
SELECT TOP 2 E.APELIDO, E.SALARIO FROM EMPREGADO E WHERE
E.SALARIO > 1500;
```

--35.Exibir a subtração das tabelas Cargo e Departamento para os campos número e nome.

```
SELECT C.NUMERO, C.NOME FROM CARGO C
```

```
EXCEPT
```

```
SELECT D.NUMERO, D.NOME FROM DEPARTAMENTO D;
```

--36.Exibir a intersecção das tabelas Cargo e Departamento para os campos número e nome.

```
SELECT C.NUMERO, C.NOME FROM CARGO C
```

```
INTERSECT
```

```
SELECT D.NUMERO, D.NOME FROM DEPARTAMENTO D;
```

--37.Exibir a união das tabelas Cargo e Departamento para os campos número e nome.

```
SELECT C.NUMERO, C.NOME FROM CARGO C
```

```
UNION
```

```
SELECT D.NUMERO, D.NOME FROM DEPARTAMENTO D;
```

--38.Exibir os empregados com os dados sobrenome, salário e número do departamento, com no mínimo uma pessoa subordinada a eles.

```
SELECT E.NUMERO, E.SOBRENOME, E.DEPARTAMENTO FROM  
EMPREGADO E WHERE E.GERENTE IS NULL AND EXISTS (SELECT 'X'  
FROM EMPREGADO X WHERE X.GERENTE = E.NUMERO);
```

--39.Exibir o somatório de Salário agrupado por Departamento e Cargo dos empregados com Número de Departamento menor que 60, exibindo os subtotais de Departamento e Cargo.

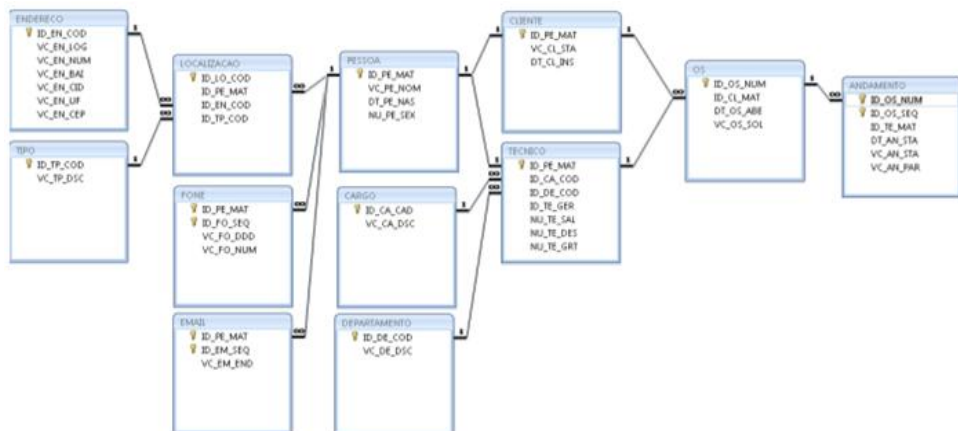
```
SELECT E.DEPARTAMENTO, E.CARGO, SUM(SALARIO) AS SOM_SAL  
FROM EMPREGADO E  
GROUP BY E.DEPARTAMENTO, E.CARGO WITH ROLLUP
```

--40.Quais os empregados possuem salário maior que o de João Abacateiro?

```
SELECT E.PRIMEIRONOME + ' ' + E.SOBRENOME AS NOME, E.SALARIO  
FROM EMPREGADO E WHERE E.SALARIO > (SELECT X.SALARIO FROM  
EMPREGADO X WHERE X.PRIMEIRONOME = 'JOÃO' AND X.SOBRENOME =  
'ABACATEIRO')
```

--Baseado no modelo abaixo responda as questões abaixo:

Figura 2.1 Modelo Entidade Relacionamento



Fonte: Próprio Autor

--41) A matrícula e o nome dos clientes com a idade superior a 45 anos.
 SELECT CLIENTE.ID_PE_MAT, PESSOA.VC_PE_NOM
 FROM PESSOA INNER JOIN CLIENTE ON PESSOA.ID_PE_MAT =
 CLIENTE.ID_PE_MAT
 WHERE (((Now()-[PESSOA].[DT_PE_NAS])>45));

--42) A matrícula dos técnicos cujos os nomes iniciem com A,F,G ou M.
 SELECT TECNICO.ID_PE_MAT
 FROM PESSOA INNER JOIN TECNICO ON
 PESSOA.ID_PE_MAT=TECNICO.ID_PE_MAT
 WHERE PESSOA.VC_PE_NOM Like 'A%' OR PESSOA.VC_PE_NOM Like 'F%'
 OR PESSOA.VC_PE_NOM Like 'G%' OR PESSOA.VC_PE_NOM Like 'M%';

--43) A quantidade de pessoas que possuem apenas uma localização.
 SELECT LOCALIZACAO.ID_PE_MAT, Count(LOCALIZACAO.ID_LO_COD) AS
 QTDE
 FROM LOCALIZACAO
 GROUP BY LOCALIZACAO.ID_PE_MAT
 HAVING Count(LOCALIZACAO.ID_LO_COD)=1;

--44) A quantidade de gestores.
 SELECT Count(*) AS QTDE_GESTORES
 FROM TECNICO
 WHERE (((TECNICO.ID_TE_GER) Is Null));

--45) O nome e o sexo M se 0 e F se 1 para os departamentos técnico e comercial.
 SELECT PESSOA.VC_PE_NOM, NU_PE_SEX

```
FROM DEPARTAMENTO INNER JOIN (PESSOA INNER JOIN TECNICO ON
PESSOA.ID_PE_MAT = TECNICO.ID_PE_MAT) ON
DEPARTAMENTO.ID_DE_COD = TECNICO.ID_DE_COD
WHERE (((DEPARTAMENTO.VC_DE_DSC)="TÉCNICO")) OR
(((DEPARTAMENTO.VC_DE_DSC)="COMERCIAL"));
```

--46) Os 2 maiores salários líquidos dos técnicos.

```
SELECT TOP 2 TECNICO.NU_TE_SAL + TECNICO.NU_TE_GRT -
TECNICO.NU_TE_DES AS LIQ
FROM TECNICO
ORDER BY 1 DESC;
```

--47) A média de salários base por departamento.

```
SELECT DEPARTAMENTO.VC_DE_DSC, Avg(TECNICO.NU_TE_SAL) AS
MédiaDeNU_TE_SAL
FROM DEPARTAMENTO INNER JOIN TECNICO ON
DEPARTAMENTO.ID_DE_COD = TECNICO.ID_DE_COD
GROUP BY DEPARTAMENTO.VC_DE_DSC;
```

--48) A quantidade de ordens que foram canceladas e encerradas do ano corrente.

```
SELECT COUNT(ANDAMENTO.ID_OS_NUM) AS QTDE
FROM ANDAMENTO
WHERE (YEAR(ANDAMENTO.DT_AN_STA)=2013) AND
(ANDAMENTO.VC_AN_STA="ENCERRADAS" OR
ANDAMENTO.VC_AN_STA="CANCELADAS");
```

--49) A matrícula e o nome dos técnicos sem cargo informado.

```
SELECT TECNICO.ID_PE_MAT, PESSOA.VC_PE_NOM
FROM PESSOA INNER JOIN TECNICO ON PESSOA.ID_PE_MAT =
TECNICO.ID_PE_MAT
WHERE (((TECNICO.ID_CA_COD) Is Null));
```

--50) A matrícula e o nome dos todos os técnicos e a descrição de seus departamentos caso exista a informação.

```
SELECT TECNICO.ID_PE_MAT, PESSOA.VC_PE_NOM,
DEPARTAMENTO.VC_DE_DSC
FROM DEPARTAMENTO RIGHT JOIN (PESSOA INNER JOIN TECNICO ON
PESSOA.ID_PE_MAT = TECNICO.ID_PE_MAT) ON
DEPARTAMENTO.ID_DE_COD = TECNICO.ID_DE_COD;
```

--51) A matrícula e o nome dos técnicos e a descrição de todos os departamentos caso exista a informação.

```
SELECT TECNICO.ID_PE_MAT, PESSOA.VC_PE_NOM,
DEPARTAMENTO.VC_DE_DSC
```

```
FROM DEPA5RTAMENTO LEFT JOIN (PESSOA RIGHT JOIN TECNICO ON  
PESSOA.ID_PE_MAT = TECNICO.ID_PE_MAT) ON  
DEPARTAMENTO.ID_DE_COD = TECNICO.ID_DE_COD;
```

--52) A soma dos salários base dos técnicos por cargo cujo o salário base seja maior que 2000.

```
SELECT CARGO.VC_CA_DSC, Sum(TECNICO.NU_TE_SAL) AS  
SomaDeNU_TE_SAL  
FROM CARGO INNER JOIN TECNICO ON  
CARGO.ID_CA_CAD=TECNICO.ID_CA_COD  
WHERE (((TECNICO.NU_TE_SAL)>2000))  
GROUP BY CARGO.VC_CA_DSC;
```

--53) A soma dos salários base dos técnicos por cargo cujo o valor da soma seja maior que 5000.

```
SELECT CARGO.VC_CA_DSC, Sum(TECNICO.NU_TE_SAL) AS  
SomaDeNU_TE_SAL  
FROM CARGO INNER JOIN TECNICO ON CARGO.ID_CA_CAD =  
TECNICO.ID_CA_COD  
GROUP BY CARGO.VC_CA_DSC  
HAVING Sum(TECNICO.NU_TE_SAL) > 5000;
```

--54) A soma dos salários líquidos dos técnicos por cargo cujo a gratificação seja maior que 1000 e a soma dos salários líquido seja maior que 3000.

```
SELECT CARGO.VC_CA_DSC,  
Sum(TECNICO.NU_TE_SAL+TECNICO.NU_TE_GRT-TECNICO.NU_TE_DES)  
AS LIQ  
FROM CARGO INNER JOIN TECNICO ON CARGO.ID_CA_CAD =  
TECNICO.ID_CA_COD  
WHERE TECNICO.NU_TE_GRT > 1000  
GROUP BY CARGO.VC_CA_DSC  
HAVING Sum(TECNICO.NU_TE_SAL+TECNICO.NU_TE_GRT-  
TECNICO.NU_TE_DES) > 3000;
```

--55) Criar índice que não permita duplicidade para nomes dos cargos.

```
CREATE UNIQUE INDEX IX01 ON CARGOS (VC_CA_NOM);
```

--56) Nomes das pessoas sem duplicidade.

```
SELECT DISTINCT PESSOA.VC_PE_NOM  
FROM PESSOA;
```

--57) Quantidade de OS abertas mas não atendidas no mês corrente.

```
SELECT OS.DT_OS_ABE  
FROM OS INNER JOIN ANDAMENTO ON OS.ID_OS_NUM <>  
ANDAMENTO.ID_OS_NUM
```

```
WHERE MONTH(OS.DT_OS_ABE) = MONTH(Now());
```

--58) A matrícula, o nome e as descrições dos departamentos e cargos dos técnicos.

```
SELECT TECNICO.ID_PE_MAT, PESSOA.VC_PE_NOM,  
DEPARTAMENTO.VC_DE_DSC, CARGO.VC_CA_DSC  
FROM CARGO INNER JOIN (DEPARTAMENTO INNER JOIN (PESSOA INNER  
JOIN TECNICO ON PESSOA.ID_PE_MAT = TECNICO.ID_PE_MAT) ON  
DEPARTAMENTO.ID_DE_COD = TECNICO.ID_DE_COD) ON  
CARGO.ID_CA_CAD = TECNICO.ID_CA_COD;
```

--59) A matrícula, o nome, os e-mails e fones dos clientes.

```
SELECT CLIENTE.ID_PE_MAT, PESSOA.VC_PE_NOM, FONE.VC_FO_NUM,  
EMAIL.VC_EM_END  
FROM ((PESSOA INNER JOIN CLIENTE ON PESSOA.ID_PE_MAT =  
CLIENTE.ID_PE_MAT) INNER JOIN FONE ON PESSOA.ID_PE_MAT =  
FONE.ID_PE_MAT) INNER JOIN EMAIL ON PESSOA.ID_PE_MAT =  
EMAIL.ID_PE_MAT;
```

--60) A matrícula, o nome, o fone, o e-mail do cliente da solicitação de OS=15.

```
SELECT CLIENTE.ID_PE_MAT, PESSOA.VC_PE_NOM, FONE.VC_FO_NUM,  
EMAIL.VC_EM_END  
FROM ((PESSOA INNER JOIN (CLIENTE INNER JOIN OS ON  
CLIENTE.ID_PE_MAT = OS.ID_CL_MAT) ON PESSOA.ID_PE_MAT =  
CLIENTE.ID_PE_MAT) INNER JOIN EMAIL ON PESSOA.ID_PE_MAT =  
EMAIL.ID_PE_MAT) INNER JOIN FONE ON PESSOA.ID_PE_MAT =  
FONE.ID_PE_MAT  
WHERE (((OS.ID_OS_NUM)=15));
```

--61) O parecer da solicitação de número 25.

```
SELECT ANDAMENTO.ID_OS_NUM, ANDAMENTO.VC_AN_PAR  
FROM ANDAMENTO  
WHERE (((ANDAMENTO.ID_OS_NUM)=25));
```

--62) O andamento da solicitação de número 30.

```
SELECT ANDAMENTO.*  
FROM ANDAMENTO  
WHERE (((ANDAMENTO.ID_OS_NUM)=30));
```

--63) A(s) matrícula(s) e nome(s) do(s) técnico(s) que possuem o maior salário.

```
SELECT TECNICO.ID_PE_MAT, PESSOA.VC_PE_NOM  
FROM PESSOA INNER JOIN TECNICO ON PESSOA.ID_PE_MAT =  
TECNICO.ID_PE_MAT  
WHERE NU_TE_SAL = (SELECT MAX(NU_TE_SAL) FROM TECNICO);
```

--64) O(s) nome(s) do(s) técnico(s) que mais realizaram atendimentos.

```
SELECT TECNICO.ID_PE_MAT, PESSOA.VC_PE_NOM  
FROM PESSOA INNER JOIN TECNICO ON PESSOA.ID_PE_MAT =  
TECNICO.ID_PE_MAT  
WHERE TECNICO.ID_PE_MAT IN (SELECT TOP 2 TECNICO.ID_PE_MAT  
FROM TECNICO INNER JOIN ANDAMENTO ON TECNICO.ID_PE_MAT =  
ANDAMENTO.ID_TE_MAT  
GROUP BY TECNICO.ID_PE_MAT  
ORDER BY Count(ANDAMENTO.ID_OS_NUM) DESC);
```

--65) A quantidade de caracteres existentes no nome das pessoas.

```
SELECT LEN(PESSOA.VC_PE_NOM) AS QTDE  
FROM PESSOA;
```

--66) A união dos cargos e departamentos ordenado por descrição decrescente.

```
SELECT ID_CA_CAD, VC_CA_DSC  
FROM CARGO  
UNION SELECT ID_DE_COD, VC_DE_DSC  
FROM DEPARTAMENTO;
```

--67) O menor salário líquido pago por departamento.

```
SELECT DEPARTAMENTO.VC_DE_DSC,  
Min(TECNICO.NU_TE_SAL+TECNICO.NU_TE_GRT-TECNICO.NU_TE_DES)  
AS LIQ  
FROM DEPARTAMENTO INNER JOIN TECNICO ON  
DEPARTAMENTO.ID_DE_COD = TECNICO.ID_DE_COD  
GROUP BY DEPARTAMENTO.VC_DE_DSC;
```

--68) O maior salário líquido pago por departamento e por cargo.

```
SELECT DEPARTAMENTO.VC_DE_DSC, CARGO.VC_CA_DSC,  
Max([TECNICO].[NU_TE_SAL]+[TECNICO].[NU_TE_GRT]-  
[TECNICO].[NU_TE_DES]) AS LIQ  
FROM CARGO INNER JOIN (DEPARTAMENTO INNER JOIN TECNICO ON  
DEPARTAMENTO.ID_DE_COD = TECNICO.ID_DE_COD) ON  
CARGO.ID_CA_CAD = TECNICO.ID_CA_COD  
GROUP BY DEPARTAMENTO.VC_DE_DSC, CARGO.VC_CA_DSC;
```

3.1 Apresentações

BD. Disponível em < <http://www.slideshare.net/alvarofpinheiro/fundamentos-de-banco-de-dados-relacionais> >

3.2 Aulas

Script da Aula de 29 Abril 2014. Disponível em < <http://goo.gl/dsYCwi> >

3.3 Ferramentas

Administração Firebird. Disponível em < <http://goo.gl/HHWsKC> >

Administração MySQL. Disponível em < <http://goo.gl/Vc02nO> >

Administração Oracle. Disponível em < <http://goo.gl/Vc02nO> >

Administração Postgre. Disponível em < <http://goo.gl/MVKicO> >

Administração SQLServer. Disponível em < <http://goo.gl/P01RTZ> >

Administração SQLServer Studio. Disponível em < <http://goo.gl/4sn8XZ> >

HeidiSQL. Disponível em < <http://www.heidisql.com/download.php> >

Instalação Firebird. Disponível em < <http://goo.gl/QcnhKu> >

Instalação MySQL. Disponível em < <http://goo.gl/vLIhvo> >

Instalação Oracle. Disponível em < <http://goo.gl/d8V6Qt> >

Instalação Postgre. Disponível em < <http://goo.gl/3e79Qj> >

Instalação SQLServer. Disponível em < <http://goo.gl/K7IPPI> >

Modelagem DBDesigner. Disponível em < <http://goo.gl/mG2VLh> >

Modelagem ERWin. Disponível em < <http://goo.gl/MFtjLc> >

MySQL Installer. Disponível em < <http://goo.gl/1hkP2> >

MySQL Reference Manual. Disponível em < <http://goo.gl/G6SAbX> >

MySQLFront. Disponível em < <http://goo.gl/nehqN> >

3.4 Sugestões de Leitura

Introdução a Sistemas de Banco de Dados. Disponível em < <http://goo.gl/7w6bjia> >

Mastering Oracle SQL. Disponível em < <http://goo.gl/DwV3VP> >

SQL Guia Prático. Disponível em < <http://goo.gl/qcYxz8> >

4.1 Referências Web

Aulas de SQL. Disponível em < <http://goo.gl/BbpyDJ> >

Exemplos de Utilização dos Comandos SQL. Disponível em < <http://goo.gl/QSRtlv> >

Gramática BNF para SQL ANSI. Disponível em < <http://savage.net.au/SQL/> >

Guia de Referência do MySQL. Disponível em < <http://goo.gl/Rnmphi> >

Guia de Referência do Postgre. Disponível em < <http://goo.gl/5tUdFF> >

Guia de Referência do SQLServer. Disponível em < <http://goo.gl/01O84d> >

Lista de Exercícios de SQL. Disponível em < <http://goo.gl/9zxrFU> >

Manual de Referência do MySQL. Disponível em < <http://goo.gl/eFX0kj> >

Padrão da Linguagem SQL. Disponível em < <http://goo.gl/xwKvG1> >

4.2 Referências Bibliográficas

BEIGHLEY, Lynn. Use a Cabeça SQL. 1ªed. Rio de Janeiro: Alta Books, 2008.

CARVALHO, Rogério. SQL Guia Prático. 2ªed. Rio de Janeiro: Brasport, 2007.

DATE, Christopher. Introdução a Sistemas de Banco de Dados. 8ªed. Rio de Janeiro: Campus, 2004.

HEUSER, Carlos. Projeto de Banco de Dados. 6ªed. Porto Alegre: Bookman, 2009.

MISTRY, Ross. Introducing Microsoft SQL Server 2008 R2. Washington: Microsoft Press, 2014.

OLIVEIRA, Haley. Postgresql 8 Referencia. The PostgreSQL Global Development Group, 2005.

OLIVEIRA, Haley. Postgresql 8 Tutorial. The PostgreSQL Global Development Group, 2005.

ORACLE. MySQL 5.5 Reference Guide, 2011.

VALADARES, Fabrício. SQL dos Conceitos às Consultas Complexas. Ciência Moderna, 2009.

Livros da série Fundamentos da Engenharia de Software



Fundament
os da
Engenharia
de
Software:
Conceitos
Básicos é
uma
coleânea
de

disciplinas que integradas servem para fundamentar o entendimento da construção de projetos de software com qualidade, isto é, baseado em processos maduros e reconhecidos pela comunidade tecnológica. O objetivo deste livro é fornecer ao leitor as bases necessárias para o desenvolvimento de aplicações sejam Desktop, Web ou Mobile. Iniciando a leitura na Teoria da Computação, passando por Processos, Linguagens, Bancos de Dados e finalizando com Sistemas de Informação e Colaboração. Este livro pode ser lido capítulo a capítulo ou somente a disciplina desejada, pois sua elaboração consiste na compilação das disciplinas fundamentais da Engenharia de Software que são independentes, mas ao mesmo tempo se integram objetivando o desenvolvimento de aplicações.



Introdução à
Banco de
Dados. Neste
são
abordados
os conceitos
básicos de
bancos de
dados e
seus

sistemas gerenciadores, mas com o foco na arquitetura relacional, porque ainda hoje o mercado faz uso em larga escala desses bancos de dados, mesmo que o paradigma predominante seja o orientado a objetos e que, já existam a um bom tempo bancos orientados a objeto, até mesmo os bancos objetos-relacionais que são um híbrido entre essas duas arquiteturas, o que predomina ainda é o relacional, assim, este material é focado na linguagem de consulta estruturada para os SGBD-Rs do mercado, com foco na comparação de cinco dos mais utilizados bancos relacionais, os quais são: Oracle, SQLServer, MySQL, SQLBase e Interbase.



Este livro é
sobre
processos
de
desenvolvim
ento de
software,
evidenciand
o a
necessidade

de qualidade na construção de sistemas, conceituando a diferença entre desenvolvimento Adhoc e com processo. Para isso é realizado a introdução à engenharia de requisitos abordando as técnicas para a elicitação de requisitos que forneçam subsídios necessários para uma construção de software com maior qualidade, enfatizando a necessidade de se aplicar na construção de qualquer sistema as técnicas de análise e modelagem, evidenciando o uso da linguagem da Linguagem de Modelagem Unificada (UML) para diagramar um projeto de software, explicando a necessidade do uso de modelos na construção, entrando com detalhes na análise orientada a objetos, com o objetivo de explorar os seus conceitos de requisitos e modelagem integrados. Este material é finalizado com a introdução à medidas de esforço de desenvolvimento, técnica necessária para responder as perguntas básicas de qualquer desenvolvimento: Qual o prazo e custo? E para responder a essas questões é abordado o uso da métrica análise de ponto de função.



Este livro aborda os sistemas que são classificados como informação, a exemplo, sistemas de apoio a

decisão, sistemas estratégicos, sistemas gerenciais e sistemas transacionais. A produção deste material que compõe o volume 4 da coleção Fundamentos da Engenharia de Software é resultado da compilação das aulas produzidas nas disciplinas que compõem os capítulos deste livro.



A motivação deste livro é exemplificar os conceitos de Padrões de Projetos utilizando a linguagem de programação

o Java, sendo a construção uma compilação das aulas produzidas com o intuito de facilitar o entendimento do assunto abordando os seguintes temas: Paradigma Orientado a Objetos que introduz o leitor nos conceitos do POO; Linguagem de Modelagem Unificada para apresentar a simbologia UML dos conceitos de POO; Linguagem de Programação Java apresentando essa poderosa linguagem de programação orientada a objetos para exemplificar os padrões de projeto; e Padrões de Projetos que neste livro aborda os mais referenciados nas academias, sendo eles o GRASP e GoF.



Este livro é o resultado do uso da ferramenta MS Project da Microsoft utilizada na aplicação dos conceitos de

gestão de projetos do PMBOK com as premissas da engenharia de testes para aquisição de qualidade nos produtos de software.



Este livro aborda basicamente e os conceitos básicos de programação o como autômatos, tipos de

linguagens, princípios dos compiladores, paradigmas de desenvolvimento e lógica de programação.



Este livro introduz nas tecnologias Web abordando os conceitos básicos para desenvolvim

ento para Internet com a apresentação da plataforma Dot Net e exibindo dicas de codificação para a linguagem de marcação ASPX, para a linguagem de script mais utilizada pelos navegadores o JavaScript com exemplos de CSS e principalmente dicas de código para a linguagem de programação CSharp e de banco de dados SQL com foco no SQLServer.

