

Oracle 12c: Fundamentals I

- SQL e SQL Developer

TOR12CF1
Junho/2015

Apostila desenvolvida especialmente para a Target Informática Ltda.
Sua cópia ou reprodução é expressamente proibida.

Sumário

1. Introdução	1
Objetivos.....	2
Ciclo de Vida do Desenvolvimento de Sistemas.....	3
Análise e Estratégia	3
Design	3
Construção e Documentação	3
Transição	3
Produção	4
Armazenamento de Dados em Diferentes Mídias.....	5
Armazenando Informações	5
Conceito de Banco de Dados Relacional	6
Modelo Relacional.....	6
Componentes do Modelo de Dados Relacional.....	6
Definição de Banco de Dados Relacional	7
Banco de Dados Relacional	7
Modelos de Dados.....	8
Propósito dos Modelos.....	8
Modelo Entidade-Relacionamento	9
Modelo ER	9
Benefícios do Modelo ER	9
Componentes Chaves.....	9
Convenções do Modelo Entidade-Relacionamento	10
Entidades.....	10
Atributos.....	10
Relacionamentos	10
Identificadores Únicos.....	11
Terminologia Utilizada em Bancos de Dados Relacionais.....	12
Relacionando Múltiplas Tabelas.....	14
Diretrizes para Chaves Primárias e Chaves Estrangeiras.....	14
Propriedades de um Banco de Dados Relacional.....	15
Comunicando com um SGDB utilizando SQL	16
Structured Query Language - SQL.....	16
Sistema de Gerenciamento de Banco de Dados.....	17
Conheça o Mundo Oracle	18
Oracle Database	18
Oracle Application Express (APEX).....	18
Oracle Application Server.....	18
Oracle Designer	18
Oracle Developer.....	18
Oracle Discover	19
Produtos Oracle.....	19
Portal de Tecnologia Oracle - OTN	19

Oracle12c:.....	20
Solução Oracle.....	21
Comandos SQL.....	22
Tabelas Utilizadas no Curso.....	24
Exercícios – 1	26
Espaço para anotações	27
2...Introdução ao comando SELECT	28
Objetivos.....	29
Características do Comando SQL SELECT.....	30
Comando SELECT Básico.....	31
Escrevendo Comandos SQL.....	32
Executando Comandos SQL no SQL Developer	32
Executando Comandos SQL no SQL*PLUS	32
Selecionando todas as Colunas.....	33
Selecionando todas as Colunas e todas as Linhas.....	33
Selecionando Colunas Específicas.....	34
Selecionando Colunas Específicas e todas as Linhas	34
Padrões de Cabeçalho de Colunas.....	36
Oracle SQL Developer.....	37
Expressões Aritméticas	38
Operadores Aritméticos.....	38
Utilizando Operadores Aritméticos.....	39
Precedência dos Operadores	40
Prioridade de Precedência.....	40
Precedência utilizando Parênteses.....	41
Definindo um Valor Nulo.....	42
Valores nulos	42
Valores Nulos em Expressões Aritméticas.....	43
Definindo um Alias de Coluna	44
Utilizando Alias de Colunas	45
Operador de Concatenação.....	46
Strings de Caracteres Literais	47
Operador alternativo para aspas (Alternative Quote operator)	49
Linhas Duplicadas.....	50
Eliminando Linhas Duplicadas	51
Interação entre SQL e SQL*Plus	52
SQL e SQL*Plus	52
SQL e SQL Developer.....	52
Características do SQL.....	53
Características do SQL*Plus.....	53
Características do SQL*Developer	55
Visão Geral do SQL*Plus	56
SQL*Plus	56
Visão Geral do SQL Developer.....	57

SQL Developer	57
Conectando com o SQL*Plus	58
Conectando com o SQL Developer.....	59
Criando uma conexão no SQL Developer	59
Propriedades da conexão no SQL Developer	61
Utilizando o SQL Developer.....	62
Executando um comando SQL no SQL Developer	62
Executando um Script de comandos SQL e SQL*PLUS no SQL Developer.....	62
Utilizando o Histórico de Comandos no SQL Developer	62
Voltando e Avançando Comandos no SQL Developer utilizando teclas.....	62
Exibindo a Estrutura de Tabelas no SQL*PLUS	63
Exibindo a Estrutura de Tabelas no SQL Developer.....	64
Tipos de Dados.....	65
Principais Comandos de Arquivo do SQL*Plus.....	67
Exercícios – 2	68
Espaço para anotações	69
3...Restringindo e Ordenando Dados.....	70
Objetivos.....	71
Limitando as Linhas Selecionadas	72
Utilizando a Cláusula WHERE	73
Strings de Caractere e Datas	74
Operadores de Comparação	75
Utilizando os Operadores de Comparação	76
Outros Operadores de Comparação	77
Operador BETWEEN.....	78
Operador IN.....	79
Operador LIKE.....	80
Combinando Padrões de Caractere	81
A Opção ESCAPE	81
Operador IS NULL	82
Operadores Lógicos.....	83
Operador AND.....	84
Combinações de Resultados com o Operador AND.....	84
Operador OR.....	85
Combinações de Resultados com o Operador OR.....	85
Operador NOT	86
Combinações de Resultados com o Operador NOT	86
Regras de Precedência	87
Utilizando Parênteses para Alterar a Prioridade.....	89
Cláusula ORDER BY	90
Classificando em Ordem Descendente.....	91
Ordenação Padrão dos Dados	91
Invertendo a Ordem Padrão	91
Ordenando pelo Alias de Coluna.....	92

Ordenando pela posição numérica da coluna.....	93
Ordenando por Múltiplas Colunas.....	94
Exercícios – 3	95
Espaço para anotações	96
4....Funções Single Row, Funções de Conversão e Expressões de Condição	97
Objetivos.....	98
Funções SQL Single Row.....	99
Tipos de Funções SQL.....	100
Funções do Tipo Single-Row	100
Funções do Tipo Multiple-Row	100
Funções do Tipo Single-Row.....	101
Características de Funções Tipo Single-Row.....	101
Funções single-row.....	102
Funções de Caracteres.....	103
Funções de Caracteres (continuação).....	104
Funções de Conversão entre Maiúsculas/Minúsculas.....	105
Utilizando Funções de Conversão entre Maiúsculas e Minúsculas	106
Funções de Manipulação de Caracteres	107
Utilizando as Funções de Manipulação de Caracteres	108
Funções Numéricas.....	109
Utilizando a Função ROUND	110
Utilizando a Função TRUNC.....	111
Utilizando a Função MOD	112
Trabalhando com Datas	113
Formato de Data Oracle	113
SYSDATE	113
DUAL.....	113
Formato Padrão de Datas.....	114
Cálculos com Datas.....	115
Utilizando Operadores Aritméticos com Datas	116
Funções de Data	117
Utilizando Funções de Data	118
Arredondando e truncando datas	119
Funções de Conversão.....	120
Conversão Implícita de Tipos de Dados	120
Conversão Explícita de Tipos de Dados	122
Função TO_CHAR com Datas	123
Exibindo uma Data em um Formato Específico	123
Elementos de Formatação de Datas	124
Formatos de Hora	124
Outros Formatos	124
Sufixos podem Influenciar a Exibição de Números	125
Utilizando a Função TO_CHAR com Datas	126
Função TO_CHAR com Números	127

Elementos de Formatação de Números	127
Utilizando a Função TO_CHAR com Números.....	129
Diretrizes	129
Funções TO_NUMBER e TO_DATE.....	130
Utilizando a Função CAST	131
Função NVL.....	132
Função NVL2	133
Utilizando a Função NVL e NVL2	134
Utilizando a Função NULLIF	135
Utilizando a Função COALESCE.....	136
Uso de CASE no SELECT	137
Função DECODE.....	139
Utilizando a Função DECODE.....	140
Aninhando Funções.....	141
Exercícios – 4	142
Espaço para anotações	143
5....Exibindo Dados a Partir de Múltiplas Tabelas	144
Objetivos.....	145
Obtendo Dados a Partir de Múltiplas Tabelas.....	146
O que é um Join?	147
Diretrizes	147
Produto Cartesiano	148
Gerando um Produto Cartesiano.....	149
Tipos de Joins.....	150
O que é um Equijoin?.....	151
Recuperando Registros com Equijoins	152
Qualificando Nomes de Colunas Ambíguos.....	153
Condições Adicionais de Pesquisa com o Operador AND	154
Utilizando Alias de Tabela	155
Diretrizes:	155
Relacionando várias Tabelas.....	156
Non-Equijoins	157
Recuperando Registros com Non-Equijoins.....	158
Outer Joins	159
Recuperando Registros sem Correspondência Direta Utilizando Outer Joins	160
Utilizando Outer Joins.....	161
Restrições do Outer Join	161
Self Joins	162
Exercícios – 5	163
Espaço para anotações	164
6....Utilizando Funções de Grupo e Formando Grupos.....	165
Objetivos.....	166
O que são Funções de Grupo?	167

Tipos de Funções de Grupo.....	168
Utilizando Funções de Grupo.....	169
Diretrizes para Utilização de Funções de Grupo	169
Utilizando as Funções AVG e SUM.....	170
Utilizando as Funções MIN e MAX.....	171
Utilizando a Função COUNT.....	172
Funções de Grupo e Valores Nulos.....	174
Criando Grupos de Dados.....	175
Criando Grupos de Dados: Cláusula GROUP BY	176
Diretrizes	176
Utilizando a Cláusula GROUP BY	177
Agrupando por mais de uma coluna ou expressões.....	179
Grupos Dentro de Grupos	179
Utilizando a Cláusula GROUP BY em Múltiplas Colunas.....	180
Grupos Dentro de Grupos	180
Consultas Ilegais Utilizando Funções de Grupo	181
Cláusula Having.....	183
Selecionando Grupos utilizando a cláusula Having	184
Aninhando Funções de Grupo.....	186
Exercícios – 6.....	187
Espaço para anotações	188
7....Variáveis de Substituição e Variáveis de ambiente do SQL*Plus	189
Objetivos.....	190
Variáveis de Substituição.....	191
Utilizando Variáveis de Substituição com (&).....	192
Utilizando o Comando SET VERIFY	193
No SQLPlus.....	193
Valores Caractere e Data com Variáveis de Substituição	194
Especificando Nomes de Colunas, Expressões e Textos em Tempo de Execução.....	195
Utilizando Variáveis de Substituição com (&&).....	196
Definindo Variáveis.....	197
O Comando ACCEPT	198
Utilizando o Comando ACCEPT.....	199
Comandos DEFINE e UNDEFINE	200
Utilizando o Comando DEFINE e UNDEFINE.....	201
Variáveis de Ambiente do SQL*Plus	202
Variáveis do Comando SET	203
Exercícios – 7	204
Espaço para anotações	205
8....Sub-consultas	206
Objetivos.....	207
Utilizando uma Sub-consulta para Resolver um Problema	208
Sub-consultas.....	209

Utilizando uma Sub-consulta.....	210
Diretrizes para Utilização de Sub-consultas	211
Tipos de Sub-consultas	212
Sub-consultas Single-Row.....	213
Múltiplas Sub-consultas Single-Row.....	214
Utilizando Funções de Grupo em uma Sub-consulta.....	215
Utilizando a cláusula Sub-consultas na cláusula HAVING.....	216
Erros utilizando Operador single row	217
Operador single row utilizado com uma Sub-consulta que não retorna nenhuma linha. .	218
Sub-consultas do Tipo Multiple-Row.....	219
Utilizando o Operador ANY em Sub-consultas Multiple-Row.....	220
Utilizando o Operador ALL em Sub-consultas Multiple-Row	221
Sub-consultas Multiple-Column	222
Utilizando Sub-consultas Multiple-Column.....	223
Utilizando uma Sub-consulta na Cláusula FROM	224
Cuidado com Sub-consultas que retornam NULL.....	225
Exercícios – 8.....	226
Espaço para anotações	227
9....Operadores SET	228
Objetivos.....	229
Operadores SET.....	230
União – UNION.....	231
Utilizando vários operadores SET	233
Precedência.....	233
Interseção - INTERSECT.....	235
Diferença - MINUS	237
Exercícios – 9.....	239
Espaço para anotações	240
10. Manipulando Dados	241
Objetivos.....	242
Linguagem de Manipulação de Dados.....	243
Comando INSERT	244
Inserindo Novas Linhas	245
Inserindo Linhas com Valores Nulos	246
Inserindo Valores Especiais.....	247
Inserindo Valores de Data Específicos	248
Inserindo Valores Utilizando Variáveis de Substituição	249
Criando um Scripts SQL com Prompts Customizados	250
INSERT utilizando uma sub-consulta	251
Comando UPDATE	252
Alterando Linhas em uma Tabela	253
UPDATE utilizando uma sub-consulta	254
Atualizando Linhas: Erro de Constraint de Integridade.....	255

Comando DELETE	256
Removendo Linhas de uma Tabela	257
DELETE utilizando uma sub-consulta	258
Removendo Linhas: Erro de Constraint de Integridade	259
Transações de Banco de Dados	260
Tipos de Transações	260
Quando uma Transação Inicia?	260
Quando uma Transação Termina?	260
Vantagens do COMMIT e ROLLBACK	261
Controlando Transações	262
Processamento Implícito de Transações	263
Falhas do Sistema	263
Situação dos Dados Antes do COMMIT ou ROLLBACK	264
Situação dos Dados Antes do COMMIT ou ROLLBACK	264
Situação dos Dados Após o COMMIT	265
Efetivando os Dados	266
Situação dos Dados Após o ROLLBACK	267
Utilizando Savepoints	268
Rollback ao Nível de Comando	269
Leitura Consistente	270
Implementação de Leitura Consistente	271
Lock	272
O que São Locks?	272
Como o Oracle Efetua o Lock dos Dados	272
SELECT FOR UPDATE	273
SELECT FOR UPDATE utilizando mais de uma tabela	274
Exercícios – 10	275
Espaço para anotações	278
11. Criando e Gerenciando Tabelas	279
Objetivos	280
Objetos do Banco de Dados	281
Estruturas de Tabelas Oracle	281
Convenções de Nomes	282
Diretrizes de Nomenclatura	282
Comando CREATE TABLE	283
Opção DEFAULT	284
Criando Tabelas	285
Consultando o Dicionário de Dados	286
Tipos de Dados	287
Criando uma Tabela Utilizando uma Sub-consulta	289
Diretrizes	289
Criando uma Tabela a Partir de uma sub-consulta	290
Comando ALTER TABLE	291
Adicionando uma Coluna	292

Diretrizes para Adicionar uma Coluna.....	292
Modificando uma Coluna	293
Diretrizes	293
Removendo uma Coluna	294
Diretrizes	294
Renomeando uma Coluna.....	295
Diretrizes	295
ALTER TABLE READY ONLY	296
Diretrizes	296
ALTER TABLE READY READ WRITE	297
Diretrizes	297
Renomeando uma Tabela	298
Diretrizes	298
Renomeando um Objeto	299
Diretrizes	299
Truncando uma Tabela.....	300
Adicionando Comentários para Tabelas e Colunas	301
Exercícios – 11.....	302
Espaço para anotações	303
12. Implementando Constraints	304
Objetivos.....	305
O Que são Constraints?.....	306
Constraints de Integridade de Dados.....	306
Diretrizes para Constraints	307
Constraints podem ser definidas constraints em dois níveis diferentes.....	307
Constraint NOT NULL.....	308
Constraint PRIMARY KEY	310
Constraint UNIQUE KEY.....	312
Constraint FOREIGN KEY	314
Palavras Chave de Constraints FOREIGN KEY.....	316
Constraint CHECK.....	317
Adicionando uma Constraint	318
Diretrizes	318
Removendo uma Constraint.....	319
Desabilitando Constraints	320
Diretrizes	320
Habilitando Constraints.....	321
Diretrizes	321
Visualizando Constraints.....	322
Visualizando as Colunas Associadas com Constraints.....	323
Exercícios – 12.....	324
Espaço para anotações	325
13. Criando Visões	326

Objetivos.....	327
O que é uma Visão?.....	328
Porquê Utilizar Visões?	329
Vantagens de Visões.....	329
Visões Simples e Visões Complexas	330
Criando uma Visão.....	331
Diretrizes para Criação de Visões.....	332
Efetuando consultas utilizando uma Visão	333
Consultando as Visões existentes.....	334
Visões no Dicionário de Dados	334
Acesso aos Dados em Visões.....	334
Modificando uma Visão	335
Criando uma Visão Complexa.....	336
Removendo uma Visão	337
Regras para Executar Operações DML em uma Visão	338
Removendo linhas através de Uma Visão.....	338
Atualizando linhas através de Uma Visão	338
Inserindo linhas através de Uma Visão	338
Utilizando a Cláusula WITH CHECK OPTION	338
Impedindo Operações DML em Visões.....	340
Exercícios – 13.....	341
Espaço para anotações	342
14. Criando Sequências, Índices e Sinônimos	343
Objetivos.....	344
O que é uma Sequence?	345
Comando CREATE SEQUENCE	346
Criando uma Sequence	347
Consultando as Sequences definidas.....	348
Pseudocolunas NEXTVAL e CURRVAL.....	349
Regras para Utilizar NEXTVAL e CURRVAL	349
Utilizando uma Sequence.....	350
Mantendo em Memória os Valores da Sequence.....	350
Previna Falhas na Sequencia.....	350
Visualizando o Próximo Valor Disponível da Sequence sem Incrementá-lo.....	350
Modificando uma Sequence.....	351
Diretrizes para Modificar uma Sequence.....	352
Removendo uma Sequence	353
O que é um Índice?.....	354
Como os Índices são Criados?	355
Criando um Índice	356
Diretrizes para a Criação de Índices.....	357
Mais nem sempre é Melhor	357
Quando Criar um Índice	357
Quando não Criar um Índice.....	357

Consultando os Índices	358
Removendo um Índice.....	359
Sinônimos.....	360
Sintaxe:.....	360
Diretrizes	360
Criando e Removendo Sinônimos.....	361
Removendo um Sinônimo	361
Exercícios – 14.....	363
Espaço para anotações	364
Apêndice 1 – Comandos do SQL*Plus.....	365
Objetivos.....	366
Comandos de Edição do SQL*Plus.....	367
Comandos de Formatação do SQL*Plus	368
Obtendo Resultados mais Legíveis.....	368
Diretrizes	368
Comando COLUMN.....	369
Utilizando o Comando COLUMN	370
Máscaras do Comando COLUMN	371
Utilizando o Comando BREAK	372
Utilizando os Comandos TTITLE e BTITLE.....	373
Criando um Arquivo de Script para Executar um Relatório	374
Passos para Criar um Arquivo de Script	374
Diretrizes	374
Relatório de Exemplo.....	375
Espaço para anotações	376
Apêndice 2 – Soluções dos Exercícios.....	377
Soluções Exercícios 2 – Introdução ao comando SELECT utilizando o SQL*PLUS e o Oracle SQL Developer	378
Soluções Exercícios 3 – Restringindo e Ordenando Dados.....	380
Soluções Exercícios 4 – Funções Single Row, Funções de Conversão e Expressões de Condição.....	382
Soluções Exercícios 5 – Exibindo Dados a Partir de Múltiplas Tabelas.....	384
Soluções Exercícios 6 – Utilizando Funções de Grupo e Formando Grupos.....	386
Soluções Exercícios 7 – Variáveis de Substituição e Variáveis de ambiente do SQL*Plus ..	388
Soluções Exercícios 8 – Sub-consultas.....	389
Soluções Exercícios 9 – Operadores SET	391
Soluções Exercícios 10 – Manipulando Dados.....	392
Soluções Exercícios 11– Criando e Gerenciando Tabelas	396
Soluções Exercícios 12 – Implementando Constraints.....	398
Soluções Exercícios 13 – Criando Visões.....	399
Soluções Exercícios 14 – Outros Objetos do Banco de Dados	401

1. Introdução

Objetivos

- Discutir os aspectos teóricos e físicos de um Banco de Dados;
- Descrever a implementação Oracle;
- Descrever como SQL é utilizado nos produtos Oracle;
- Detalhar as tabelas utilizadas no curso.

Ciclo de Vida do Desenvolvimento de Sistemas

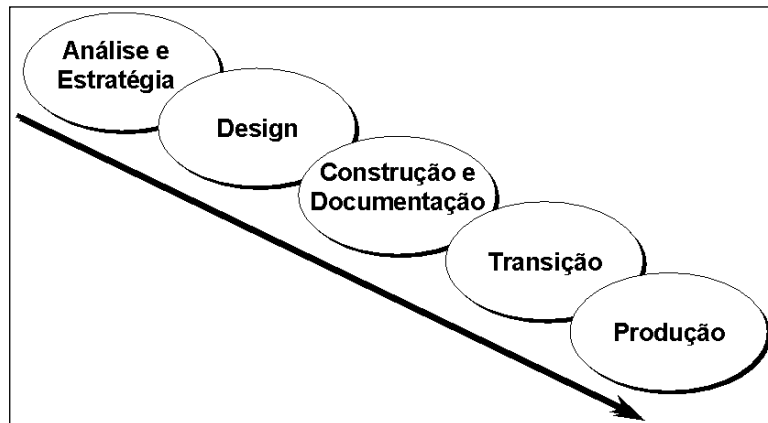


Figura 1-1: Ciclo de vida do desenvolvimento de sistemas

Do conceito à produção, você pode desenvolver um sistema baseado na tecnologia de banco de dados utilizando as técnicas do ciclo de vida do desenvolvimento de sistemas que contém várias fases:

Análise e Estratégia

- Estudo e análise das necessidades de negócio. Entrevistas com usuários e gerentes para identificar as necessidades de informações.
- Construção dos modelos do sistema. Conversão das narrativas em uma representação gráfica de informações e regras de negócio necessárias. Confirmação e refinamento do modelo com os analistas e peritos.

Design

- Projete o banco de dados baseado no modelo desenvolvido na fase de análise e estratégia.

Construção e Documentação

- Construa o protótipo do sistema. Escreva e execute os comandos para criar as tabelas e os objetos de suporte para o banco de dados.
- Desenvolva a documentação de usuário, texto de ajuda, e os manuais de operação para auxiliar na utilização do sistema.

Transição

- Refine o protótipo. Coloque a aplicação em produção após os testes dos usuários, convertendo os dados existentes. Efetue quaisquer modificações necessárias.

Produção

- Coloque o sistema em produção, monitorando sua operação e desempenho, efetuando melhorias e refinamentos necessários.

Armazenamento de Dados em Diferentes Mídias

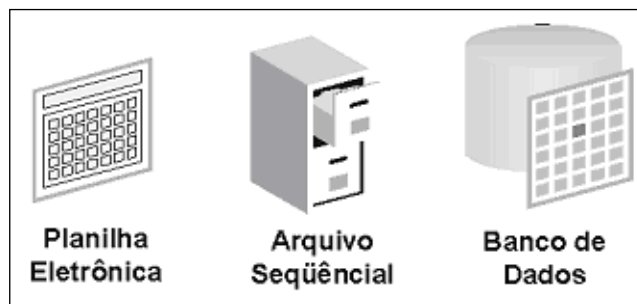


Figura 1-2: Armazenamento de dados em diferentes mídias

Armazenando Informações

Toda organização possui necessidades de informação. Uma biblioteca mantém uma lista de sócios, livros, datas de entrega e multas. Uma empresa precisa armazenar informações sobre empregados, departamentos e salários. Estes pedaços de informação são chamados de dados.

Organizações podem armazenar dados em várias mídias e em formatos diferentes. Por exemplo, uma cópia física de um documento pode estar armazenada em um arquivo, ou em dados armazenados em planilhas eletrônicas ou em bancos de dados.

Um banco de dados é um conjunto organizado de informações.

Para administrar um banco de dados, você precisa de sistemas de gerenciamento de banco de dados (SGDB). Um SGDB é um programa que armazena, recupera, e modifica os dados sempre que solicitado. Existem quatro tipos principais de bancos de dados: hierárquico, de rede, relacional, e mais recentemente relacional de objeto.

Conceito de Banco de Dados Relacional

Modelo Relacional

Os princípios do modelo relacional foram esboçados pela primeira vez pelo Dr. E. F. Codd por volta de junho de 1970 em um *paper* intitulado "Um Modelo Relacional de Dados para Grandes Bancos de Dados Compartilhados". Neste *paper*, o Dr. Codd propôs o modelo relacional para sistemas de banco de dados.

Os modelos mais populares usados naquele momento eram hierárquicos e de rede, ou até mesmo arquivo de dados com estruturas simples. Sistemas de gerenciamento de banco de dados relacionais (SGDB) logo tornaram-se populares, especialmente devido a sua facilidade de uso e flexibilidade na estrutura. Além disso, haviam vários vendedores inovadores, como a Oracle que completou o SGBD com um conjunto poderoso de ferramentas para o desenvolvimento de aplicações e produtos para o usuário, provendo uma solução total.

Componentes do Modelo de Dados Relacional

- Conjunto de objetos ou relações que armazenam os dados
- Conjunto de operadores que podem agir nas relações para produzir outras relações
- Integridade de dados para precisão e consistência

Definição de Banco de Dados Relacional

Banco de Dados Relacional

Um banco de dados relacional utiliza relações ou tabelas bidimensionais para armazenar informações.

Por exemplo, você poderia querer armazenar informações sobre todos os empregados de sua companhia. Em um banco de dados relacional, você cria várias tabelas para armazenar pedaços diferentes de informação sobre seus empregados, como uma tabela de empregado, uma tabela de departamento, e uma tabela de salário.

Modelos de Dados

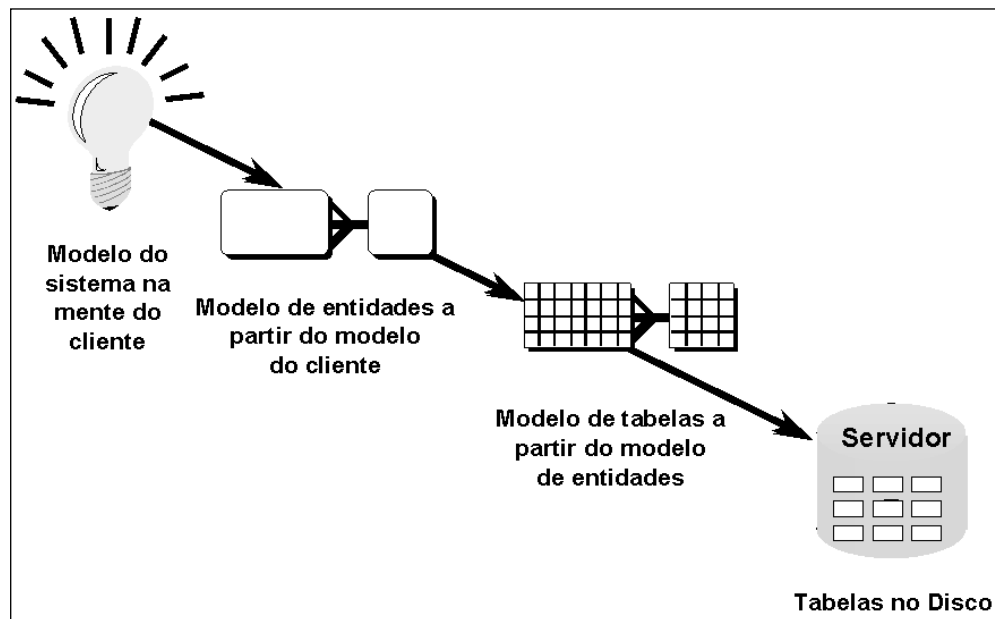


Figura 1-3: Modelo de dados

Modelos são à base de design. Engenheiros constroem um modelo de um carro para trabalhar qualquer detalhe antes de colocá-lo em produção. Da mesma maneira, projetistas de sistemas desenvolvem modelos para explorar ideias e melhorar a compreensão do design de um banco de dados.

Propósito dos Modelos

- Modelos ajudam a disseminar os conceitos nas mentes das pessoas. Eles podem ser utilizados para os seguintes propósitos:
 - Comunicar
 - Categorizar
 - Descrever
 - Especificar
 - Investigar
 - Evoluir
 - Analisar
 - Imitar

O objetivo é produzir um modelo que reúna vários destes propósitos e que possa ao mesmo tempo ser entendido por um usuário final e conter detalhes suficientes para um desenvolvedor construir um sistema de banco de dados.

Modelo Entidade-Relacionamento

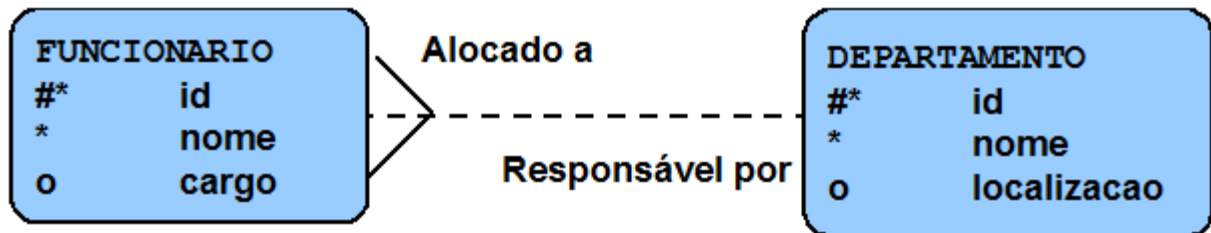


Figura 1-4: Modelo entidade-relacionamento

Modelo ER

Em um sistema efetivo, os dados são divididos em categorias ou entidades distintas. Um modelo entidade-relacionamento (ER) é uma ilustração de várias entidades em um negócio e as relações entre elas. Um modelo ER é derivado de especificações empresariais ou narrativas e construído durante a fase de análise do ciclo de vida do desenvolvimento de sistemas. Modelos ER separam as informações necessárias por um negócio a partir das atividades executadas dentro deste. Embora as empresas possam mudar suas atividades, o tipo de informação tende a permanecer constante. Portanto, as estruturas de dados também tendem a ser constantes.

Benefícios do Modelo ER

- Informação de documentos para a organização em um formato claro, preciso
- Provê um quadro claro do escopo de exigência de informação
- Provê um mapa facilitador para o design do banco de dados

Componentes Chaves

- **Entidade:** algo que possui significado sobre o qual informações precisam ser conhecidas. Exemplos: contratos, clientes, departamentos, empregados e pedidos.
- **Atributo:** descreve ou qualifica uma entidade. Por exemplo, para a entidade de contrato, os atributos seriam o número do contrato, a data de compra, desconto, total e assim por diante. Cada um dos atributos pode ser requerido ou opcional. Este estado é chamado obrigatoriedade.
- **Relacionamento:** associação nomeada entre entidades mostrando a obrigatoriedade e grau. Exemplos: clientes e contratos; empregados e departamentos; pedidos e itens.

Convenções do Modelo Entidade-Relacionamento

Entidades

Para representar uma entidade em um modelo, utilize as seguintes convenções:

- Caixa com qualquer dimensão.
- Nome da entidade no singular, único e em maiúsculo.
- Sinônimo opcional em maiúsculo dentro de parênteses: ().

Atributos

Para representar um atributo em um modelo, siga as seguintes convenções:

- Utilize nomes no singular em minúsculo.
- Marque os atributos que compõe o identificador único principal da entidade com um símbolo #
- Marque atributos obrigatórios, ou valores que devem ser conhecidos, com um asterisco: *
- Marque atributos opcionais, ou valores que podem ser conhecidos, com o símbolo o

Relacionamentos

Cada lado da relação contém:

- Um nome. Por exemplo: **ensinado por** ou **designado para**
- Uma obrigatoriedade: **deve ser** ou **pode ser**
- Um grau: ambos **um e somente um** ou um dos lados **um ou mais**

Nota: O termo cardinalidade é um sinônimo para o termo grau.

Cada entidade de origem {pode ser | deve ser} nome da relação {um e somente um | um ou mais} entidade de destino.

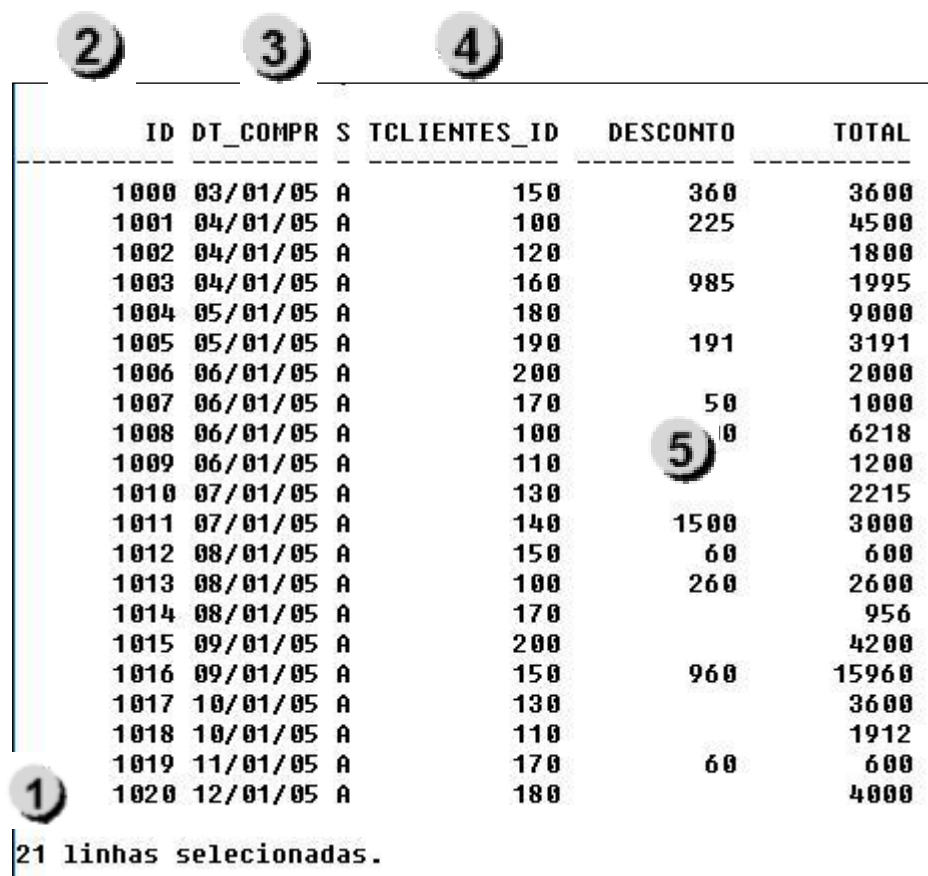
Símbolo	Descrição o relacionamento
Linha pontilhada	Elemento opcional indicando "pode ser"
Linha sólida	Elemento obrigatório indicando "deve ser"
Pé de "galinha"	Elemento de grau indicando "um ou mais"
Linha simples	Elemento de grau indicando "um e somente um"

Identificadores Únicos

Um identificador único (UID) é qualquer combinação de atributos ou relações, ou ambos, que servem para distinguir ocorrências distintas de uma mesma entidade. Cada ocorrência da entidade deve ser exclusivamente identificada.

- Marque cada atributo que é parte do UID com o símbolo: #
- Marque UIDs secundários com o símbolo entre parênteses: (#)

Terminologia Utilizada em Bancos de Dados Relacionais



ID	DT_COMPR	S	TCLIENTES_ID	DESCONTO	TOTAL
1000	03/01/05	A	150	360	3600
1001	04/01/05	A	100	225	4500
1002	04/01/05	A	120		1800
1003	04/01/05	A	160	985	1995
1004	05/01/05	A	180		9000
1005	05/01/05	A	190	191	3191
1006	06/01/05	A	200		2000
1007	06/01/05	A	170	50	1000
1008	06/01/05	A	100	0	6218
1009	06/01/05	A	110		1200
1010	07/01/05	A	130		2215
1011	07/01/05	A	140	1500	3000
1012	08/01/05	A	150	60	600
1013	08/01/05	A	100	260	2600
1014	08/01/05	A	170		956
1015	09/01/05	A	200		4200
1016	09/01/05	A	150	960	15960
1017	10/01/05	A	130		3600
1018	10/01/05	A	110		1912
1019	11/01/05	A	170	60	600
1020	12/01/05	A	180		4000

21 linhas selecionadas.

Figura 1-6: Terminologia de banco de dados relacional

Um banco de dados relacional pode conter uma ou muitas tabelas. Uma tabela é uma estrutura básica de armazenamento em um SGDB. Uma tabela armazena todos os dados necessários sobre algo no mundo real, por exemplo: empregados, faturas ou clientes.

A figura acima mostra o conteúdo da tabela TCONTRATOS. Os números indicam o seguinte:

1. Linha ou tupla representando todos os dados necessários para um único contrato. Cada linha de uma tabela deve ser identificada por uma chave primária que não permite linhas duplicadas. A ordem das linhas é insignificante; especifica-se a ordem das linhas quando os dados são recuperados.
2. Coluna ou atributo que contém o número do contrato, sendo também a chave primária. Esta coluna identifica cada contrato da tabela TCONTRATOS. Uma chave primária deve sempre conter um valor.
3. Coluna que não é um valor chave. Uma coluna representa um tipo de dado em uma tabela; no exemplo, a data de compra de todos os contratos. A ordem das colunas é insignificante quando armazenando dados; especifica-se a ordem das colunas quando os dados são recuperados.
4. Coluna que contém o número do cliente, sendo também uma chave estrangeira. Uma chave estrangeira é uma coluna que define como uma tabela

se relaciona com outra. Uma chave estrangeira se referencia a uma chave primária ou uma chave única em outra tabela. No exemplo, `TCLIENTES_ID` identifica um único cliente da tabela `TCLIENTES`.

5. Um campo pode ser encontrado na interseção de uma linha com uma coluna. Somente um valor poderá existir nesta interseção.

Nota: Valores nulos serão discutidos mais adiante em capítulos subsequentes.

Relacionando Múltiplas Tabelas

Cada tabela contém dados que descrevem exatamente uma entidade. Por exemplo, a tabela TCONTRATOS contém informações sobre contratos.

Devido aos dados de diferentes entidades serem armazenados em diferentes tabelas, pode ser necessário combinar duas ou mais tabelas para responder uma questão específica. Por exemplo, pode-se querer saber a data de compra do contrato onde é definido por um cliente. Neste caso, você precisa de informações da tabela TCONTRATOS (que contém os dados sobre os contratos) e da tabela TCLIENTES (que contém informações sobre os clientes). Um SGDB permite relacionar os dados em uma tabela com os dados contidos em outra utilizando-se as chaves estrangeiras. Uma chave estrangeira é uma coluna ou um conjunto de colunas que se referem a uma chave primária da mesma tabela ou de outra.

A facilidade para relacionar dados em uma tabela com dados em outra tabela permite a organização das informações em unidades separadas. Os dados de contratos podem ser mantidos logicamente distintos de dados de clientes armazenando-os em tabelas separadas.

Diretrizes para Chaves Primárias e Chaves Estrangeiras

- Nenhum valor duplicado é permitido em uma chave primária.
- Geralmente, chaves primárias não podem ser alteradas.
- Chaves estrangeiras estão baseadas em valores de dados e são puramente lógicas, não sendo físicas ou ponteiros.
- Um valor de chave estrangeira tem que existir como um valor de chave primária ou chave única, ou então ser NULO.

Propriedades de um Banco de Dados Relacional

Em um banco de dados relacional, você não especifica o caminho de acesso para as tabelas, e também não necessita saber como os dados estão organizados fisicamente.

Para acessar o banco de dados, você executa comandos SQL (structured query language), que é a linguagem padrão para operação sobre bancos de dados relacionais, definida pelo American National Standards Institute (ANSI). Esta linguagem possui um grande conjunto de operadores para dividir e combinar as relações. O banco de dados pode ser modificado utilizando-se comandos SQL.

Comunicando com um SGDB utilizando SQL



Figura 1-7: Comunicando com um banco de dados relacional utilizando SQL

Structured Query Language - SQL

A linguagem SQL lhe permite comunicar com o servidor e possui as seguintes vantagens:

- Eficiência.
- Fácil de aprender e utilizar.
- Completa funcionalidade. A linguagem SQL permite definir, recuperar, e manipular dados em tabelas.

Sistema de Gerenciamento de Banco de Dados

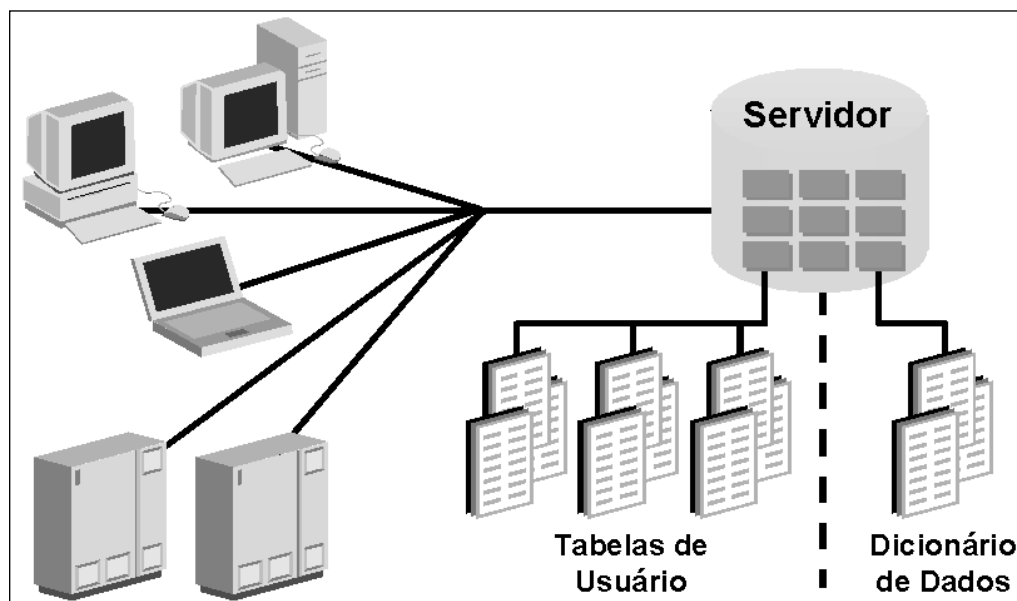


Figura 1-8: SGDB - Sistema de gerenciamento de banco de dados

A Oracle fornece um SGDB flexível. Suas características o permitem armazenar e administrar dados com todas as vantagens da estrutura relacional mais as vantagens do PL/SQL, uma ferramenta que lhe proporciona a capacidade para armazenar e executar unidades de programas.

O Servidor oferece para os usuários opções para recuperar dados baseado em técnicas de otimização. Inclui características de segurança que controlam como o banco de dados é acessado e utilizado. Outra característica é a consistência e a proteção dos dados através de mecanismos de bloqueio (lock).

As aplicações Oracle podem rodar no mesmo computador onde encontra-se o Servidor Oracle. Entretanto, você pode rodar as aplicações em um sistema local para o usuário e pode rodar o Servidor Oracle em outro sistema (arquitetura cliente-servidor). Neste ambiente cliente-servidor, pode ser utilizado um grande número de recursos de computação. Por exemplo, uma aplicação de reserva de linhas aéreas pode rodar em um computador pessoal cliente ao mesmo tempo em que obtém acesso aos dados de voos que são administrados convenientemente por um Servidor Oracle em um computador central.

Conheça o Mundo Oracle

Oracle Database

Sistema Gerenciador de Banco de Dados, com grande performance e satisfação de Clientes. O Oracle Server possibilita, além do armazenamento dos dados convencionais, a total gerência sobre os dados não estruturados como documentos, mensagens, páginas de internet, filmes, etc. Tudo isto com a melhor performance, estabilidade e escalabilidade em ambientes que vão desde computadores pessoais (Oracle Express Edition) até grandes mainframes corporativos (Oracle Enterprise Edition).

O ORACLE DATABASE 12c oferece a forma mais segura, escalável e rentável para a gerência de todas as informações de sua corporação nas seguintes versões.

- Oracle Database Enterprise Edition.
- Oracle Database Standard Edition.
- Oracle Database Standard Edition One.
- Oracle Database Express Edition (atualmente na versão 11g).

Oracle Application Express (APEX)

Permite a construção de aplicações utilizando somente um browser. Possibilita projetar, desenvolver e implantar paliativos baseados na tecnologia de banco de dados com uma interface simples e robusta, gerando sistemas e páginas WEB.

Oracle Application Server

Primeiro a trazer a confiabilidade para transações na Web, este produto permite, de uma forma extremamente segura, construir aplicações robustas e escaláveis em ambiente WEB. Aplicações totalmente dinâmicas com acesso transacional ao banco de dados são facilmente construídas com o Oracle Application Server.

Oracle Designer

O Designer é uma ferramenta de altíssima produtividade para o desenvolvimento de aplicações em equipes. Um poderoso repositório compartilhado associado a diagramas gráficos permitem a criação de complexos aplicativos automaticamente gerados para diversas linguagens. Tudo isto sem a necessidade da codificação manual e com altos índices de produtividade, redução de custos e tempo de desenvolvimento.

Oracle Developer

Ferramenta de quarta geração, multiplataforma para o desenvolvimento rápido de aplicações cliente / servidor e Internet. Orientação a Objetos, múltiplos ambientes e altíssima integração com bancos de dados permitem a construção de aplicações robustas e escaláveis de forma simultânea tanto para ambiente Web quanto Cliente / Servidor.

Oracle Discover

O Discoverer é a ferramenta perfeita para análise, construção de relatórios, consultas e gráficos voltados para o usuário final. O Discoverer combina o poder e a flexibilidade da tecnologia de acesso a dados com uma interface intuitiva e de fácil uso, simplificando dramaticamente o processo de criação de gráficos, realização de consultas e de relatórios os quais podem também ser executados via Internet. Ferramenta ideal para mineração de dados em datawarehouses corporativos.

Produtos Oracle

Conheça toda a linha de produtos e serviços da Oracle em:

<http://www.oracle.com>

Portal de Tecnologia Oracle - OTN

Conheça o portal de tecnologia Oracle em:

<http://www.oracle.com/technology/index.html>

Oracle12c:

Introduz uma nova arquitetura *multitenant* que é uma arquitetura em que uma única instância do software do banco de dados atende a vários clientes (bancos de dados).

Esta arquitetura permite consolidar vários bancos de dados de forma rápida e permite gerenciá-los como um serviço de nuvem (cloud – daí o “c” da versão).

O Oracle12c também traz o processamento de dados *in-memory*, provendo capacidades de entrega e desempenho analítico avançado.

Inovações adicionais do banco de dados oferecem novos níveis de eficiência, segurança, performance e disponibilidade.

Solução Oracle

O principal produto da Oracle é o seu sistema de gerenciamento de banco de dados, incluindo o Servidor Oracle e várias ferramentas com o objetivo de auxiliar os usuários na manutenção, monitoramento, e uso dos dados. O dicionário de dados Oracle é um dos componentes mais importantes do Servidor. Consiste em um conjunto de tabelas e visões que proveem uma referência somente de leitura para o banco de dados.

O SGDB administra tarefas como as seguintes:

- Administração do armazenamento e definição dos dados
- Controla e restringe o acesso aos dados e a concorrência
- Provê procedimentos de cópia e restauração dos dados
- Interpreta comandos SQL e PL/SQL

Nota: PL/SQL é uma linguagem procedural desenvolvida pela Oracle que estende as definições do SQL adicionando-lhe lógica de aplicação.

Comandos SQL e PL/SQL são utilizados por todos os programas e usuários para acessar e manipular dados armazenados no banco de dados Oracle. Utilizando programas aplicativos é possível ter acesso ao banco de dados sem utilizar SQL ou PL/SQL diretamente, porque você pode apertar um botão ou selecionar um "check box", por exemplo, mas os aplicativos implicitamente utilizam SQL ou PL/SQL para executar sua solicitação.

O SQL Developer é uma nova ferramenta disponibilizada pela Oracle que interpreta comandos SQL e PL/SQL para serem executados no servidor. Trata-se de uma ferramenta gráfica que torna mais amigável a utilização, bem como o gerenciamento de dados do banco de dados.

SQL*Plus, por sua vez, é uma ferramenta Oracle em modo texto que interage com o banco de dados em mais baixo nível, sem qualquer formatação e tratamento dos dados e possui sua própria linguagem de comandos.

A Oracle oferece também uma enorme variedade de interfaces gráficas de usuário (GUI) em ferramentas voltadas para a construção de aplicações empresariais como também um grande conjunto de softwares aplicativos para muitas áreas de negócio e indústrias.

Nota: Maiores informações sobre o dicionário de dados da Oracle serão vistas em capítulos posteriores.

Comandos SQL

SELECT	Recuperação de dados
INSERT UPDATE DELETE	Linguagem de manipulação de dados (DML)
CREATE ALTER DROP RENAME TRUNCATE	Linguagem de definição de estruturas de dados (DDL)
COMMIT ROLLBACK SAVEPOINT	Controle de transações
GRANT REVOKE	Linguagem de controle de dados (DCL)

Figura 1-9: Comandos SQL

O SQL Oracle segue os padrões aceitos pela indústria. A Oracle assegura compatibilidade futura com a evolução destes padrões adicionando chaves próprias ao SQL definido pelos comitês. Comitês aceitos pela indústria são o Instituto de Padrões Americano (ANSI) e a Organização de Padrões Internacional (ISO). ANSI e ISO adotaram o SQL como a linguagem padrão para bancos de dados relacionais.

Comando	Descrição
SELECT	Recupera dados a partir do banco de dados
INSERT UPDATE DELETE	Respectivamente insere novas linhas, modifica registros existentes e remove linhas não desejadas a partir de tabelas do banco de dados. Coletivamente conhecidos como linguagem de manipulação de dados – DML (data manipulation language)
CREATE ALTER DROP RENAME TRUNCATE	Cria, modifica e remove estruturas de dados a partir de tabelas. Coletivamente conhecidos como linguagem de definição de dados – DDL (data definition language)
COMMIT ROLLBACK SAVEPOINT	Administra as mudanças efetuadas por comandos DML. As alterações nos dados podem ser agrupadas em transações lógicas.
GRANT REVOKE	Fornece ou remove direitos de acesso ao banco de dados e as estruturas nele definidas. Coletivamente conhecidos como linguagem de controle de dados – DCL (data control language)

Tabela 1-2: Comandos SQL

Tabelas Utilizadas no Curso

Serão utilizadas as seguintes tabelas no curso:

Tabela de Clientes: TCLIENTES

Define as principais informações sobre os clientes. Sua chave primária é o campo ID.

Column Name	Data Type	Nullable
ID	NUMBER(6,0)	No
NOME	VARCHAR2(35 BYTE)	No
DT_NASCIMENTO	DATE	Yes
ENDERECO	VARCHAR2(40 BYTE)	Yes
CIDADE	VARCHAR2(30 BYTE)	Yes
ESTADO	VARCHAR2(2 BYTE)	Yes
CEP	VARCHAR2(8 BYTE)	Yes
TELEFONE	VARCHAR2(10 BYTE)	Yes
COMENTARIOS	VARCHAR2(1000 BYTE)	Yes

Figura 1-10: Tabela TCLIENTES

Tabela de Contratos: TCONTRATOS

Define os contratos de cursos realizados pelos clientes. Sua chave primária é o campo ID. Possui uma chave estrangeira obrigatória com a tabela de clientes a partir do campo TCLIENTES_ID.

Column Name	Data Type	Nullable
ID	NUMBER(4,0)	No
DT_COMPRA	DATE	Yes
STATUS	VARCHAR2(1 BYTE)	Yes
TCLIENTES_ID	NUMBER(6,0)	No
DESCONTO	NUMBER(7,2)	Yes
TOTAL	NUMBER(10,2)	Yes

Figura 1-11: Tabela TCONTRATOS

Tabela de Itens: TITENS

Define os itens pertencentes a um contrato, especificando a quantidade e o total. Sua chave primária é o conjunto de campos TCONTRATOS_ID (referente à tabela de contratos) e SEQ (que especifica a sequência de um item de um contrato).

Column Name	Data Type	Nullable
TCONTRATOS_ID	NUMBER(4,0)	No
SEQ	NUMBER(4,0)	No
TCURSOS_ID	NUMBER(6,0)	Yes
QTDE	NUMBER(3,0)	Yes
TOTAL	NUMBER(8,2)	Yes

Figura 1-12: Tabela TITENS

Tabela de Cursos: TCURSOS

Define informações gerais sobre os cursos oferecidos (produto de um item de um contrato). Sua chave primária é ID, mas nesta tabela temos um auto-relacionamento para especificar os pré-requisitos de cada curso, ou seja, uma chave estrangeira opcional para a própria tabela de cursos (o campo PRE_REQUISITO referenciando ID).

Column Name	Data Type	Nullable
ID	NUMBER(4,0)	No
NOME	VARCHAR2(100 BYTE)	Yes
COD_TRG	VARCHAR2(8 BYTE)	Yes
PRECO	NUMBER(7,2)	Yes
CARGA_HORARIA	NUMBER(3,0)	Yes
DT_CRIACAO	DATE	Yes
PRE_REQUISITO	NUMBER(4,0)	Yes

Figura 1-13: Tabela TCURSOS

Tabela de Descontos: TDESCONTOS

Define a classe de descontos aplicados nos contratos. Sua chave primária é CLASSE.

Column Name	Data Type	Nullable
CLASSE	VARCHAR2(2 BYTE)	No
BASE_INFERIOR	NUMBER(7,2)	Yes
BASE_SUPERIOR	NUMBER(7,2)	Yes

Figura 1-14: Tabela TDESCONTOS

Nota: as tabelas e seus dados serão criadas no momento dos exercícios.

Exercícios – 1

Acompanhar o instrutor para criação do ambiente utilizado no curso.

1. Identificar os programas necessários para criação: PuTTY, SQL Developer e SQL*Plus.
2. Executar o PuTTY:
Host Name (or IP address): **10.0.10.30** – open
Login as: **oracle**
Password: **oracle**
3. Acessar a pasta dos scripts:
Comando: **cd /u01/setup/fundamentals1**
4. Iniciar uma sessão do SQL*Plus:
Comando: **sqlplus**
Usuário: **sys@ora12c as sysdba**
Senha: **oracle**
5. Executar o script que cria o usuário: o nome do usuário será distribuído pelo instrutor; a senha é igual ao nome do usuário informado:

```
SQL> @criausuario;
```

6. Ainda no SQL*Plus, conectar-se com o usuário criado:

```
SQL> conn usuario@ora12c;
```

7. Executar o script que cria o ambiente do curso (tabelas e dados):

```
SQL> @setup;
```

8. Desconectar do banco de dados e sair do SQL*Plus:

```
SQL> disco  
SQL> exit
```


Espaço para anotações

2. Introdução ao comando SELECT

Objetivos

- Listar as características do comando SQL SELECT;
- Executar um comando SELECT básico;
- Conhecer o Oracle SQL Developer;
- Conhecer o SQL*Plus.

Características do Comando SQL SELECT

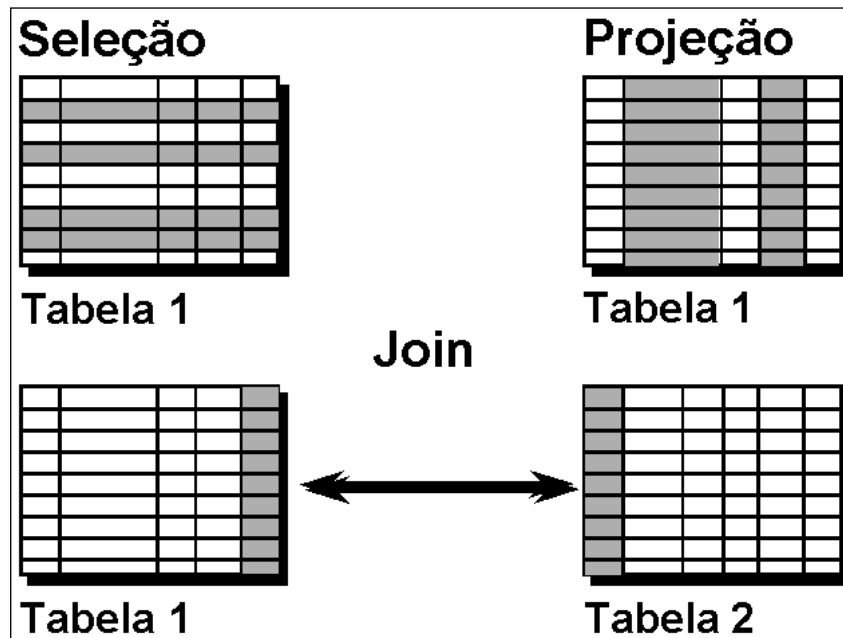


Figura 2-1: Características do comando SQL SELECT

Um comando SELECT recupera informações a partir do banco de dados. Usando um comando SELECT você pode fazer o seguinte:

- **Seleção:** Você pode usar a capacidade de seleção em SQL para escolher as linhas de uma tabela que você deseja recuperar através de uma consulta. Você pode usar vários critérios para seletivamente restringir as linhas que serão visualizadas.
- **Projeção:** Você pode usar a capacidade de projeção em SQL para escolher as colunas de uma tabela que você deseja recuperar através de uma consulta. Você pode escolher algumas ou todas as colunas de uma tabela, de acordo com sua necessidade.
- **Join:** Você pode usar a capacidade de JOIN em SQL para reunir dados que estão armazenados em tabelas diferentes, criando um vínculo através de colunas que ambas as tabelas compartilhem. Você aprenderá mais sobre joins em um capítulo posterior.

Comando SELECT Básico

Em sua forma mais simples, um comando SELECT deve incluir o seguinte:

- Uma cláusula SELECT que especifica as colunas a serem exibidas.
- Uma cláusula FROM que especifica as tabelas que possuem as colunas listadas na cláusula SELECT.

```
SELECT { [DISTINCT] * | column | expression [column alias],... }  
FROM table [table alias];
```

Sintaxe:

- **SELECT:** palavra chave do comando: lista de uma ou mais colunas ou expressões;
- **DISTINCT:** comando que unifica linhas repetidas;
- *****: coringa que indica todas as colunas de uma determinada tabela;
- ***column*:** nome da coluna selecionada;
- ***expression*:** expressão aritmética, concatenação de strings ou valores literais;
- ***column alias*:** título diferente para a uma determinada coluna;
- **FROM:** palavra chave para tabelas;
- ***table*:** nome da tabela origem dos dados;
- ***table alias*:** título diferente para uma tabela.

Escrevendo Comandos SQL

Seguindo regras simples e as diretrizes apresentadas abaixo, você pode construir comandos válidos que são tanto fáceis de ler quanto de editar:

- Comandos SQL não fazem distinção entre maiúsculas e minúsculas, a menos que especificado.
- Podem ser escritos em uma ou mais linhas.
- Palavras chaves (keywords) não podem ser divididas em mais de uma linha ou abreviadas.
- As diferentes cláusulas são normalmente separadas em linhas distintas para facilitar a visualização e edição do comando.
- Identações podem ser utilizadas para tornar o código mais legível.
- No SQL Developer, um comando SQL pode ser terminado ou não com (;) opcionalmente e sua interface permite a livre edição do comando.
- Dentro do SQL*Plus, um comando SQL é escrito no prompt de SQL, e as linhas subsequentes são numeradas. Isto é chamado de SQL buffer. Somente um comando pode estar no buffer de cada vez. A edição de comandos é rígida e menos amigável.
- No SQL*Plus, você necessita terminar cada comando SQL com um (;).

Executando Comandos SQL no SQL Developer

- A tecla F9 executa os comandos SQL.
- Mais de um comando pode estar na janela de edição, neste caso, separe os comandos com ponto-e-vírgula (;) ou
- Selecione o comando desejado e execute com a tecla F9.
- O prompt indica qual comando será executado.
- A tecla F5 executa blocos PL/SQL.

Executando Comandos SQL no SQL*PLUS

- Coloque um ponto-e-vírgula (;) ao término da última cláusula.
- Coloque uma barra (/) na última linha do buffer.
- Coloque uma barra (/) no prompt de SQL.
- Execute o comando RUN do SQL*Plus no prompt de SQL.

Selecionando todas as Colunas

```
SELECT *
FROM tclientes;
```

Selecionando todas as Colunas e todas as Linhas

Você pode exibir todas as colunas de dados de uma tabela colocando um asterisco (*) logo após a palavra chave SELECT. No exemplo acima, a tabela de clientes possui nove colunas: ID, NOME, DATA_NASCIMENTO, ENDERECO, CIDADE, ESTADO, CEP, TELEFONE e COMENTARIOS. A tabela possui onze linhas, uma para cada cliente.

Você também pode exibir todas as colunas da tabela listando-as depois da palavra chave SELECT. Por exemplo, o seguinte comando SQL, como no exemplo acima, também exibe todas as colunas e todas as linhas da tabela TCLIENTES:

```
SELECT id, nome, dt_nascimento, endereco, cidade,
       estado, cep, telefone, comentarios
FROM tclientes;
```

ID	NOME	DT_NASCIMENTO	ENDERECO	CIDADE	ESTADO	CEP	TELEFONE	COMENTARIOS
1	100 Amarildo Fagundes	22-DEC-76	Rua Carlos Eduardo Filho	111 São Paulo	SP	52299003	1154433267	(null)
2	110 Ana Maria Prado	03-MAR-71	Avenida Guarani 4490	Rio de Janeiro	RJ	21223444	2154462281	(null)
3	120 Antônio da Silva	10-JUL-64	Rua Carlos Dunnas 1213	São Paulo	SP	05434303	1157991238	(null)
4	130 Ramiro Antunes	19-OCT-61	Avenida Castro Alves 60	Rio de Janeiro	RJ	21938888	2145591454	(null)
5	140 Nadia Velasques	20-JAN-73	Rua Pinheiros 209	Rio de Janeiro	RJ	21935503	2144392200	(null)
6	150 Alfredo Fonseca	24-APR-70	Avenida Tapes 2831	Porto Alegre	RS	90440300	5133442874	(null)
7	160 Jânio Marques	11-MAY-65	Rua Maria da Graça 22	São Paulo	SP	54552333	1154457300	(null)
8	170 José Batista	05-FEB-69	Rua Comendador Dutra 3420	Porto Alegre	RS	90786500	5133267755	(null)
9	180 Carlos Magno	30-OCT-71	Rua Anacleto Farias 312	São Paulo	SP	05498722	1155543437	(null)
10	190 João Medeiros	15-SEP-65	Avenida das Bromélias 449	Rio de Janeiro	RJ	21333090	2145567947	(null)
11	200 Mário Cardoso	07-AUG-77	Rua Guarapari 2003	São Paulo	SP	54441003	1145567947	(null)

Selecionando Colunas Específicas

```
SELECT id, nome  
FROM tclientes;
```

	ID	NOME
1	100	Amarildo Fagundes
2	110	Ana Maria Prado
3	120	Antônio da Silva
4	130	Ramiro Antunes
5	140	Nadia Velasques
6	150	Alfredo Fonseca
7	160	Jânio Marques
8	170	José Batista
9	180	Carlos Magno
10	190	João Medeiros
11	200	Mário Cardoso

Selecionando Colunas Específicas e todas as Linhas

Você pode usar o comando SELECT para exibir colunas específicas da tabela informando os nomes das colunas, separados por vírgulas. O exemplo acima exibe todos os ids e nomes.

nomes de clientes a partir da tabela TCLIENTES.

Na cláusula SELECT, especifique as colunas que você quer ver, na ordem na qual você quer que elas sejam mostradas. Por exemplo, para exibir a data de nascimento antes do nome do cliente, você utiliza o seguinte comando:

```
SELECT dt_nascimento, nome  
FROM tclientes;
```


	DT_NASCIMENTO	NOME
1	22-DEC-76	Amarildo Fagundes
2	03-MAR-71	Ana Maria Prado
3	10-JUL-64	Antônio da Silva
4	19-OCT-61	Ramiro Antunes
5	20-JAN-73	Nadia Velasques
6	24-APR-70	Alfredo Fonseca
7	11-MAY-65	Jânio Marques
8	05-FEB-69	José Batista
9	30-OCT-71	Carlos Magno
10	15-SEP-65	João Medeiros
11	07-AUG-77	Mário Cardoso

Padrões de Cabeçalho de Colunas

Cabeçalhos e dados de colunas tipo caractere bem como cabeçalhos e dados de colunas tipo data são alinhados à esquerda. Cabeçalhos e dados de colunas numéricas são alinhados à direita.

```
SELECT id, nome, dt_nascimento  
FROM tclientes;
```

	ID	NOME	DT_NASCIMENTO
1	100	Amarildo Fagundes	22-DEC-76
2	110	Ana Maria Prado	03-MAR-71
3	120	Antônio da Silva	10-JUL-64
4	130	Ramiro Antunes	19-OCT-61
5	140	Nadia Velasques	20-JAN-73
6	150	Alfredo Fonseca	24-APR-70
7	160	Jânio Marques	11-MAY-65
8	170	José Batista	05-FEB-69
9	180	Carlos Magno	30-OCT-71
10	190	João Medeiros	15-SEP-65
11	200	Mário Cardoso	07-AUG-77

Normalmente os cabeçalhos de coluna aparecem em maiúsculas. Você pode substituir os cabeçalhos de colunas por um alias. Alias de colunas serão vistos posteriormente neste capítulo.

Oracle SQL Developer

O Oracle SQL Developer é um ambiente integrado, especializado em desenvolvimento de unidades armazenadas de programas para o banco de dados Oracle. Essa ferramenta oferece facilidade de utilização, melhor qualidade no desenvolvimento de códigos, aumento da produtividade e vantagens durante o desenvolvimento de aplicações Oracle.

É uma ferramenta gratuita com suporte para SQL e PL/SQL e recursos como descrições de objetos, assistente de código, relatórios e outros recursos sofisticados, o desenvolvedor terá mais uma série de possibilidades ao usar esse programa.

The screenshot displays the Oracle SQL Developer interface. The top menu bar includes options like Arquivo, Editar, Exibir, Navegar, Executar, Origem, Equipe, Ferramentas, Janela, and Ajuda. The left sidebar shows a tree view of database objects under 'Conexões' and 'Relatórios'. The main window is divided into two panes: 'Planilha' (Worksheet) and 'Query Builder'. The 'Query Builder' pane shows a SQL query: `select * from tclientes`. Below the query, the 'Resultado da Consulta' (Query Result) pane displays a table with 11 rows of data. The table has columns: ID, NOME, DT_NASCIMENTO, ENDEREÇO, CIDADE, ESTADO, CEP, TELEFONE, and COMENTAR.

ID	NOME	DT_NASCIMENTO	ENDEREÇO	CIDADE	ESTADO	CEP	TELEFONE	COMENTAR
1	100 Amarildo Fagundes	22-DEC-76	Rua Carlos Eduardo Filho	111 São Paulo	SP	52299003	1154433267	(null)
2	110 Ana Maria Prado	03-MAR-71	Avenida Guarani 4490	Rio de Janeiro	RJ	21223444	2154462281	(null)
3	120 Antônio da Silva	10-JUL-64	Rua Carlos Dunnas 1213	São Paulo	SP	05434303	1157991238	(null)
4	130 Ramiro Antunes	19-OCT-61	Avenida Castro Alves 60	Rio de Janeiro	RJ	21938888	2145591454	(null)
5	140 Nadia Velasques	20-JAN-73	Rua Pinheiros 209	Rio de Janeiro	RJ	21935503	2144392200	(null)
6	150 Alfredo Fonseca	24-APR-70	Avenida Tapes 2831	Porto Alegre	RS	90440300	5133442874	(null)
7	160 Jânio Marques	11-MAY-65	Rua Maria da Graça 22	São Paulo	SP	54552333	1154457300	(null)
8	170 José Batista	05-FEB-69	Rua Comendador Dutra 3420	Porto Alegre	RS	90786500	5133267755	(null)
9	180 Carlos Magno	30-OCT-71	Rua Anacleto Farias 312	São Paulo	SP	05498722	1155543437	(null)
10	190 João Medeiros	15-SEP-65	Avenida das Bromélias 449	Rio de Janeiro	RJ	21333090	2145567947	(null)
11	200 Mário Cardoso	07-AUG-77	Rua Guarapari 2003	São Paulo	SP	54441003	1145567947	(null)

Figura 2-7: Exemplo de consulta no Oracle SQL Developer

Expressões Aritméticas

Crie expressões em dados tipo NUMBER e DATE utilizando operadores aritméticos

Operador	Descrição
+	Soma
-	Subtração
*	Multiplicação
/	Divisão

Figura 2-2: Operadores aritméticos

Você pode precisar modificar a forma como os dados são exibidos, por exemplo, executando cálculos. Isto é possível através do uso de expressões aritméticas. Uma expressão aritmética pode conter nomes de colunas, valores numéricos constantes, e os operadores aritméticos.

Operadores Aritméticos

A figura acima mostra os operadores aritméticos disponíveis em SQL. Você pode usar os operadores aritméticos em qualquer cláusula de um comando SQL, exceto a cláusula FROM.

Utilizando Operadores Aritméticos

```
SELECT cod_trg, preco, preco+300  
FROM tcursos;
```

	COD_TRG	PRECO	PRECO+300
1	TO10G1	1800	2100
2	TO10G2	1800	2100
3	TF6I	1275	1575
4	TR6I	1275	1575
5	TF6I-A	956	1256
6	TSQLST	1000	1300
7	TSQLAV	1000	1300
8	TAT8I	956	1256
9	TMDPBD	800	1100
10	TDR6IM	956	1256
11	TDR6ID	956	1256

O exemplo acima utiliza o operador de adição para calcular um aumento do preço de R\$300 para todos os cursos e mostrar uma nova coluna PRECO+300 na tela.

Note que a coluna resultante PRECO+300 não é uma nova coluna da tabela TCURSOS, sendo utilizada somente na exibição. Por default, o nome da coluna nova é obtido a partir da expressão de cálculo que a gerou, neste caso, PRECO+300.

Nota: O SQL ignora espaços em branco antes e depois do operador aritmético.

Precedência dos Operadores

Prioridade de Precedência

```
1. * /
2. + -
```

As regras normais de precedência de operadores aritméticos se aplicam ao SQL.

Se uma expressão aritmética possui mais de um operador, os operadores de multiplicação e divisão são avaliados primeiro (possuindo a mesma prioridade), seguidos pelos operadores de adição e subtração (possuindo a mesma prioridade).

Quando os operadores dentro de uma expressão são da mesma prioridade, então a avaliação é feita da esquerda para direita.

Você pode usar parênteses para forçar a expressão colocada dentro deles a ser avaliada primeiro.

```
SELECT cod_trg, preco, 10*preco+preco*0.17
FROM tcursos;
```

	COD_TRG	PRECO	10*PRECO+PRECO*0.17
1	TO10G1	1800	18306
2	TO10G2	1800	18306
3	TF6I	1275	12966.75
4	TR6I	1275	12966.75
5	TF6I-A	956	9722.52
6	TSQLST	1000	10170
7	TSQLAV	1000	10170
8	TAT8I	956	9722.52
9	TMDPBD	800	8136
10	TDR6IM	956	9722.52
11	TDR6ID	956	9722.52

Observe que as multiplicações são executadas antes da adição.

Precedência utilizando Parênteses

Use parênteses para reforçar a ordem padrão de precedência e melhorar a clareza do comando. Por exemplo, a expressão acima poderia ser escrita desta forma, sem mudança no resultado: $(10*\text{preco})+(\text{preco}*0.17)$.

```
SELECT cod_trg, preco, (10*preco)+(preco*0.17)
FROM tcursos;
```

Você pode alterar as regras de precedência usando parênteses para especificar a ordem na qual devem ser executados os operadores.

```
SELECT cod_trg, preco, 10*(preco+preco)*0.17
FROM tcursos;
```

	COD_TRG	PRECO	$10*(\text{PRECO}+\text{PRECO})*0.17$
1	TO10G1	1800	6120
2	TO10G2	1800	6120
3	TF6I	1275	4335
4	TR6I	1275	4335
5	TF6I-A	956	3250.4
6	TSQLST	1000	3400
7	TSQLAV	1000	3400
8	TAT8I	956	3250.4
9	TMDPBD	800	2720
10	TDR6IM	956	3250.4
11	TDR6ID	956	3250.4

No exemplo acima, devido ao uso dos parênteses a adição recebe prioridade sobre a multiplicação, porque primeiro serão resolvidos os parênteses (de dentro para fora) e as operações dentro dos parênteses seguem a regra de prioridades.

Definindo um Valor Nulo

Valores nulos

Se uma linha não possui valor para uma coluna específica o valor para esta coluna é nulo, ou ela contém nulo.

Um valor nulo é um valor que é indisponível, não atribuído, desconhecido, ou inaplicável (ausência de valor). Um valor nulo não é o mesmo que zero ou um espaço. Zero é um número, e um espaço é um caractere.

Colunas de qualquer tipo de dado podem conter valores nulos, a menos que a coluna tenha sido definida como NOT NULL ou como PRIMARY KEY (chave primária) quando criada.

```
SELECT id, tclientes_id, desconto, total
FROM tcontratos;
```

	ID	TCLIENTES_ID	DESCONTO	TOTAL
1	1000	150	360	3600
2	1001	100	225	4500
3	1002	120	(null)	1800
4	1003	160	985	1995
5	1004	180	(null)	9000
6	1005	190	191	3191
7	1006	200	(null)	2000
8	1007	170	50	1000
9	1008	100	700	6218
10	1009	110	(null)	1200
11	1010	130	(null)	2215

Na coluna DESCONTO da tabela TCONTRATOS, você pode observar que alguns contratos possuem valores de desconto sobre o total do contrato. Um valor nulo representa o fato do cliente não possuir desconto.

Nota: no SQL Developer os campos com valores nulos são mostrados com (null), mas isto é mera formatação da ferramenta. No banco de dados, estes valores não existem.

Valores Nulos em Expressões Aritméticas

Se o valor de alguma coluna em uma expressão aritmética é nulo, o resultado da expressão também é nulo. Por exemplo, se você tentar executar uma divisão por zero, você obtém um erro. Porém, se você divide um número por nulo, o resultado é nulo ou desconhecido.

No exemplo acima, os contratos que não possuem desconto possuem o cálculo do contrato nulo. Uma vez que a coluna DESCONTO na expressão aritmética é nula, o resultado também é nulo.

```
SELECT id, desconto, total, total-desconto
FROM tcontratos;
```

	ID	DESCONTO	TOTAL	TOTAL-DESCONTO
1	1000	360	3600	3240
2	1001	225	4500	4275
3	1002	(null)	1800	(null)
4	1003	985	1995	1010
5	1004	(null)	9000	(null)
6	1005	191	3191	3000
7	1006	(null)	2000	(null)
8	1007	50	1000	950
9	1008	700	6218	5518
10	1009	(null)	1200	(null)
11	1010	(null)	2215	(null)
12	1011	1500	3000	1500
13	1012	60	600	540
14	1013	260	2600	2340
15	1014	(null)	956	(null)
16	1015	(null)	4200	(null)
17	1016	960	15960	15000
18	1017	(null)	3600	(null)
19	1018	(null)	1912	(null)
20	1019	60	600	540
21	1020	(null)	4000	(null)

Definindo um Alias de Coluna

Para exibir o resultado de uma consulta, o SQL*Plus e o SQL Developer utilizam o nome da coluna selecionada como seu cabeçalho. Em muitos casos, este título pode não ser descritivo e conseqüentemente pode ser difícil de entender. Você pode mudar o cabeçalho de uma coluna utilizando um alias (apelido) de coluna.

Especifique o alias depois da coluna na lista da cláusula SELECT utilizando um espaço como separador. Por default, cabeçalhos baseados em alias aparecem em maiúsculas. Se o alias possui espaços, caracteres especiais (como # ou \$), ou deve diferenciar maiúsculas e minúsculas, coloque o alias entre aspas duplas (" ").

Utilizando Alias de Colunas

```
SELECT id codigo, nome AS cliente  
FROM tclientes;
```

CODIGO	CLIENTE
100	Amarildo Fagundes
110	Ana Maria Prado
120	Antônio da Silva
130	Ramiro Antunes
140	Nadia Velasques
150	Alfredo Fonseca
160	Jânio Marques
170	José Batista
180	Carlos Magno
190	João Medeiros
200	Mário Cardoso

```
SELECT nome "Cliente Preferencial"  
FROM tclientes;
```

Cliente Preferencial
Amarildo Fagundes
Ana Maria Prado
Antônio da Silva
Ramiro Antunes
Nadia Velasques
Alfredo Fonseca
Jânio Marques
José Batista
Carlos Magno
João Medeiros
Mário Cardoso

O primeiro exemplo exibe o código e o nome de todos os clientes. Observe que a palavra chave opcional AS foi utilizada antes do nome do alias da coluna. O resultado da consulta seria o mesmo se a palavra chave AS fosse utilizada ou não. Observe também que o comando SQL tem os alias de coluna, id e nome em minúsculas, sendo que o resultado da consulta exibe os cabeçalhos de coluna em maiúsculas. Como mencionado anteriormente, os títulos de coluna aparecem em maiúsculas por default.

O segundo exemplo exibe o nome de todos os clientes. Uma vez que o alias "Cliente Preferencial" contém espaços, foi colocado entre aspas duplas. Observe que o cabeçalho da coluna ficou exatamente igual ao seu alias.

Operador de Concatenação

Você pode unir colunas com outras colunas, expressões aritméticas ou valores constantes para criar uma expressão de caracteres usando o operador de concatenação (||). Colunas em qualquer lado do operador são combinadas para fazer uma única coluna de saída.

```
SELECT id||nome AS "Clientes"
FROM tclientes;
```

A Z	Clientes
100	Amarildo Fagundes
110	Ana Maria Prado
120	Antônio da Silva
130	Ramiro Antunes
140	Nadia Velasques
150	Alfredo Fonseca
160	Jânio Marques
170	José Batista
180	Carlos Magno
190	João Medeiros
200	Mário Cardoso

No exemplo, são concatenadas as colunas ID e NOME, sendo que o resultado recebe o alias de "Clientes". Observe que o id do cliente e o seu nome são combinados obtendo-se uma única coluna de saída.

A palavra chave AS antes do nome do alias torna a cláusula SELECT mais legível.

Strings de Caracteres Literais

Um literal é qualquer caractere, expressão ou número incluídos na lista da cláusula SELECT que não é um nome de coluna ou um alias de coluna. Ele é impresso para cada linha retornada. Podem ser incluídas strings literais de texto no resultado da consulta e podem ser tratadas como uma coluna da lista do SELECT.

Literais do tipo data e caractere devem ser incluídas dentro de aspas simples (' '), enquanto que literais numéricas não necessitam de aspas.

```
select id || ' ' || nome "Clientes"
from tclientes;
```

Clientes
100 Amarildo Fagundes
110 Ana Maria Prado
120 Antônio da Silva
130 Ramiro Antunes
140 Nadia Velasques
150 Alfredo Fonseca
160 Jânio Marques
170 José Batista
180 Carlos Magno
190 João Medeiros
200 Mário Cardoso

```
SELECT nome || ' nasceu em ' || dt_nascimento
        AS "Nascimento do Cliente"
FROM tclientes;
```

Nascimento do Cliente
Amarildo Fagundes nasceu em 22/12/76
Ana Maria Prado nasceu em 03/03/71
Antônio da Silva nasceu em 10/07/64
Ramiro Antunes nasceu em 19/10/61
Nadia Velasques nasceu em 20/01/73
Alfredo Fonseca nasceu em 24/04/70
Jânio Marques nasceu em 11/05/65
José Batista nasceu em 05/02/69
Carlos Magno nasceu em 30/10/71
João Medeiros nasceu em 15/09/65
Mário Cardoso nasceu em 07/08/77

O exemplo acima mostra os nomes e as datas de nascimento de todos os clientes. A coluna possui o cabeçalho "Nascimento do Cliente". Observe os espaços entre as aspas simples no comando SELECT. Os espaços melhoram a visualização do resultado.

No exemplo seguinte, o id, nome e telefone de cada cliente são concatenados como um literal para dar mais significado às linhas retornadas.

```
SELECT id||': '||nome||' telefone: '||telefone CLIENTE
FROM tclientes;
```

CLIENTE
100: Amarildo Fagundes telefone: 1154433267
110: Ana Maria Prado telefone: 2154462281
120: Antônio da Silva telefone: 1157991238
130: Ramiro Antunes telefone: 2145591454
140: Nadia Velasques telefone: 2144392200
150: Alfredo Fonseca telefone: 5133442874
160: Jânio Marques telefone: 1154457300
170: José Batista telefone: 5133267755
180: Carlos Magno telefone: 1155543437
190: João Medeiros telefone: 2145567947
200: Mário Cardoso telefone: 1145567947

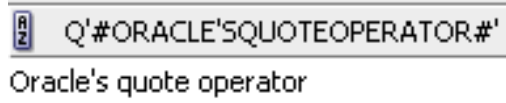
Operador alternativo para aspas (Alternative Quote operator)

- Você pode especificar um operador alternativo para aspas (alternative quote operator)
- Facilita a leitura e usabilidade

```
q'c texto a ser colocado entre aspas c'
```

Exemplo:

```
SELECT q'#Oracle's quote operator#'
FROM dual;
```


A screenshot of a SQL query result. It shows a table with one column and one row. The column header is 'Q'#ORACLE'SQUOTEOPERATOR#' and the row value is 'Oracle's quote operator'.

Q'#ORACLE'SQUOTEOPERATOR#'
Oracle's quote operator

Linhas Duplicadas

A menos que você indique o contrário, o SQL*Plus exibe os resultados de uma consulta sem eliminar as linhas duplicadas.

```
SELECT cidade CIDADES
FROM tclientes;
```

 CIDADES
São Paulo
Rio de Janeiro
São Paulo
Rio de Janeiro
Rio de Janeiro
Porto Alegre
São Paulo
Porto Alegre
São Paulo
Rio de Janeiro
São Paulo

O exemplo acima exibe todas as cidades da tabela de clientes. Observe que algumas são repetidas.

Eliminando Linhas Duplicadas

Para eliminar linhas duplicadas do resultado da consulta, inclua a palavra chave **DISTINCT** imediatamente após a palavra **SELECT**. No exemplo acima, a tabela **TCLIENTES** na verdade possui 11 linhas, mas existem somente 3 cidades diferentes na tabela.

Você pode especificar múltiplas colunas depois da palavra **DISTINCT**. O qualificador **DISTINCT** afeta todas as colunas selecionadas, e o resultado representa uma combinação distinta das colunas.

```
SELECT DISTINCT cidade
FROM   tclientes;
```

CIDADE
São Paulo
Rio de Janeiro
Porto Alegre

```
SELECT DISTINCT nome CLIENTE, cidade CIDADES
FROM   tclientes;
```

CLIENTE	CIDADES
Antônio da Silva	São Paulo
Ana Maria Prado	Rio de Janeiro
José Batista	Porto Alegre
Alfredo Fonseca	Porto Alegre
Jânio Marques	São Paulo
João Medeiros	Rio de Janeiro
Mário Cardoso	São Paulo
Carlos Magno	São Paulo
Amarildo Fagundes	São Paulo
Ramiro Antunes	Rio de Janeiro
Nadia Velasques	Rio de Janeiro

Interação entre SQL e SQL*Plus

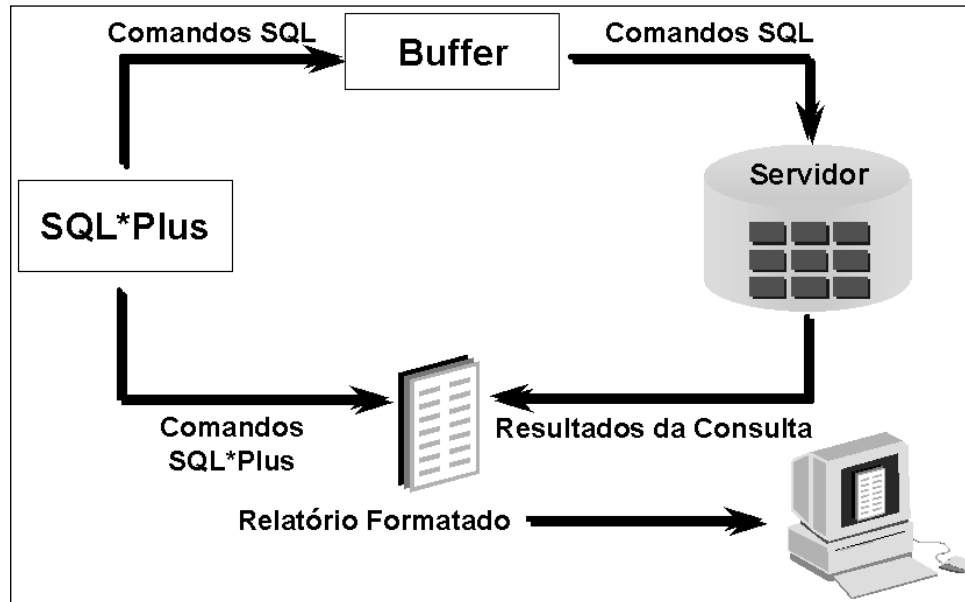


Figura 2-3: Interação entre SQL e SQL*Plus

SQL e SQL*Plus

SQL é uma linguagem de comandos para comunicação com o Servidor Oracle a partir de qualquer ferramenta ou aplicação. O SQL Oracle possui muitas extensões. Quando você entra um comando SQL, este é armazenado em uma área de memória chamada de SQL buffer e permanece lá até que você entre um novo comando.

SQL*Plus é uma ferramenta Oracle que reconhece e submete comandos SQL ao Servidor Oracle para execução e contém sua própria linguagem de comandos.

SQL e SQL Developer

O SQL Developer, assim como o SQL*Plus, é uma ferramenta para interação com o banco de dados Oracle que interpreta os comandos SQL. A interação com o banco de dados é bastante semelhante à interação do SQL*Plus (ver figura 2-3). Dentre várias diferenças, talvez a principal, é que o SQL Developer é uma Guest User Interface – GUI (interface gráfica de usuário, em tradução livre) e a própria ferramenta formata os "relatórios" ou resultados. O SQL Developer possui várias funcionalidades ao clicar de um mouse, por exemplo, mostrar estruturas de tabelas, executar comandos e scripts.

Além disso, o SQL Developer permite uma edição facilitada de comandos e scripts em uma janela similar a um editor de texto, bem como a execução de diversos SQLs de uma mesma janela.

Características do SQL

- Pode ser usado por uma grande variedade de usuários, inclusive por aqueles com pouca ou nenhuma experiência em programação;
- É uma linguagem não procedural;
- Reduz a quantidade de tempo necessária para criar e manter sistemas;
- É facilmente interpretado em tradução livre do idioma inglês para o português.

Características do SQL*Plus

- Aceita a entrada de comandos SQL a partir de arquivos;
- Possui um editor de linha para modificar comandos SQL;
- Controles de configurações de ambiente;
- Formatação do resultado de consultas em relatórios básicos;
- Acessa banco de dados locais e remotos.

A tabela abaixo compara SQL e SQL*Plus:

SQL	SQL*Plus
É uma linguagem para comunicação com o Servidor Oracle para acessar os dados	Reconhece comandos SQL e os envia ao Servidor
Está baseado no padrão SQL ANSI (American National Standards Institute)	Interface proprietária Oracle para executar comandos SQL
Manipula dados e a definição de tabelas no banco de dados	Não permite a manipulação de valores no banco de dados
Comandos armazenados no SQL buffer em uma ou mais linhas	Uma linha de cada vez; não são armazenados no SQL buffer
Não possui um caractere de continuação	Possui um hífen (-) como caractere de continuação se o comando é maior que uma linha
Não pode ser abreviado	Pode ser abreviado
Utiliza um caractere de terminação para executar o comando imediatamente	Não requer caracteres de terminação; os comandos são executados imediatamente
Utiliza funções para executar algumas formatações	Utiliza comandos para formatar os dados

Tabela 2-1: Interação entre SQL e SQL*Plus

Características do SQL Developer

- Aceita a entrada de comandos SQL a partir de arquivos, abrindo-os para edição;
- Possui um editor de texto para modificar comandos SQL;
- Controles de configurações de ambiente;
- Resultados formatados em uma grade semelhante a uma planilha;
- Impressão direta do resultado já formatado;
- Exporta a grade de resultados;
- E edição rápida dos dados;
- Acessa banco de dados locais e remotos.

Visão Geral do SQL*Plus

- Conecte ao Servidor Oracle com o SQL*Plus;
- Descreva a estrutura de tabelas;
- Edite e execute comandos SQL;
- Salve comandos SQL para arquivos e adicione comandos SQL para arquivos já existentes.
- Execute arquivos salvos.
- Carregue comandos a partir de um arquivo para o buffer para editar.

SQL*Plus

SQL*Plus é um ambiente no qual você pode fazer o seguinte:

- Executar comandos SQL para recuperar, modificar, adicionar e remover dados do banco de dados.
- Formatar, executar cálculos, armazenar e imprimir o resultado de consultas na forma de relatórios.
- Criar arquivos com scripts para armazenar comandos SQL para uso repetitivo no futuro.

Comandos SQL*Plus podem ser divididos nas seguintes categorias principais:

Categoria	Propósito
Ambiente	Afetam o comportamento geral dos comandos SQL na sessão
Formatação	Formatam os resultados das consultas
Manipulação de Arquivos	Salvam, carregam e executam arquivos de script
Execução	Enviam comandos SQL a partir do SQL buffer para o Servidor Oracle
Edição	Modificam comandos SQL no buffer
Interação	Permitem criar e passar variáveis para comandos SQL, imprimir valores de variáveis e imprimir mensagens na tela
Diversos	Existem vários comandos para conectar ao banco de dados, manipular o ambiente do SQL*Plus e mostrar as definições de colunas

Tabela 2-2: Tipos de comandos do SQL*Plus

Visão Geral do SQL Developer

- Conecte ao Servidor Oracle com o SQL Developer;
- Mostre a estrutura de tabelas ao clicar no nome organizado em uma árvore;
- Edite e execute comandos SQL a partir do editor de texto;
- Salve comandos SQL para arquivos e edite estes arquivos;
- Abra e execute arquivos salvos;
- Carregue comandos a partir de um arquivo para o editor de texto para editar.

SQL Developer

SQL Developer é um ambiente no qual você pode fazer o seguinte:

- Executar comandos SQL para recuperar, modificar, adicionar e remover dados do banco de dados.
- Utilizar a edição rápida de resultados;
- Formatar, executar cálculos, armazenar e imprimir o resultado de consultas na forma de relatórios.
- Criar arquivos com scripts para armazenar comandos SQL para uso repetitivo no futuro.

Conectando com o SQL*Plus

A forma como você executa o SQL*Plus depende do tipo de sistema operacional ou ambiente Windows que você está executando.

Para conectar em um ambiente de linha de comando:

`sqlplus [nomeusuario[/senha[@database]]]`

1º. Conecte com sua máquina.

2º. Entre o comando SQL*Plus como mostrado na figura acima.

No comando:

nomeusuario é o nome do seu usuário no banco de dados.

senha é sua senha no banco de dados; se colocada na linha de comando, ela estará visível.

@database é a string de conexão para o banco de dados.

Exemplo na linha de comando utilizando o Oracle 12c:

```
C:\Users>sqlplus target/target@local
SQL*Plus: Release 12.1.0.2.0 Production on Mon Jun 1 17:25:21 2015
Copyright (c) 1982, 2014, Oracle. All rights reserved.
Last Successful login time: Mon Jun 01 2015 16:44:16 -03:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing opt
ions
SQL>
```

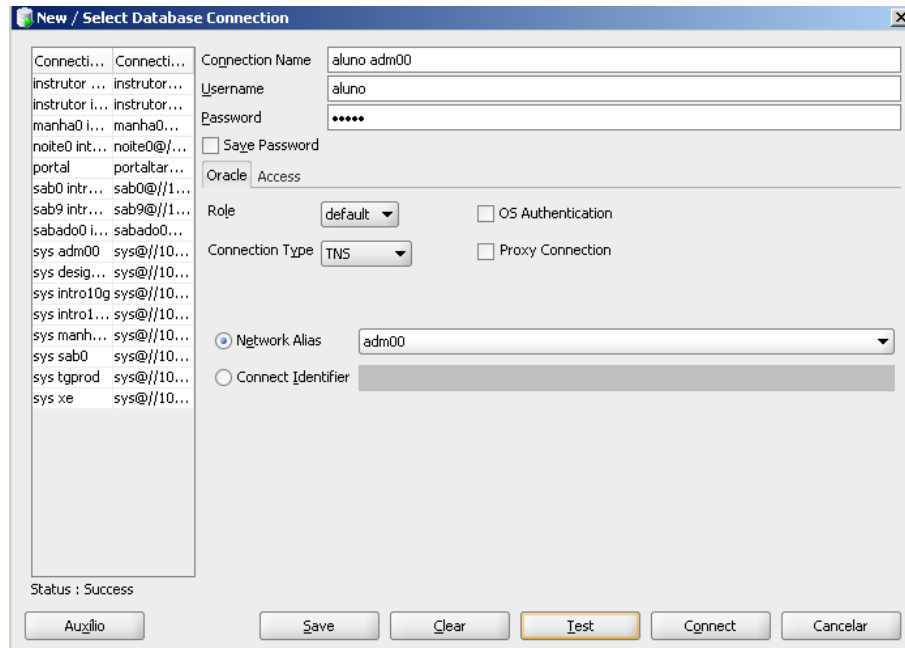
Nota: Para assegurar a integridade de sua senha, não coloque-a no prompt do sistema operacional. Ao contrário, coloque apenas seu nome de usuário e string de conexão. Entre com sua senha somente no prompt de senha do SQL*Plus. Se usuário e senha não forem informados, eles serão solicitados.

Conectando com o SQL Developer

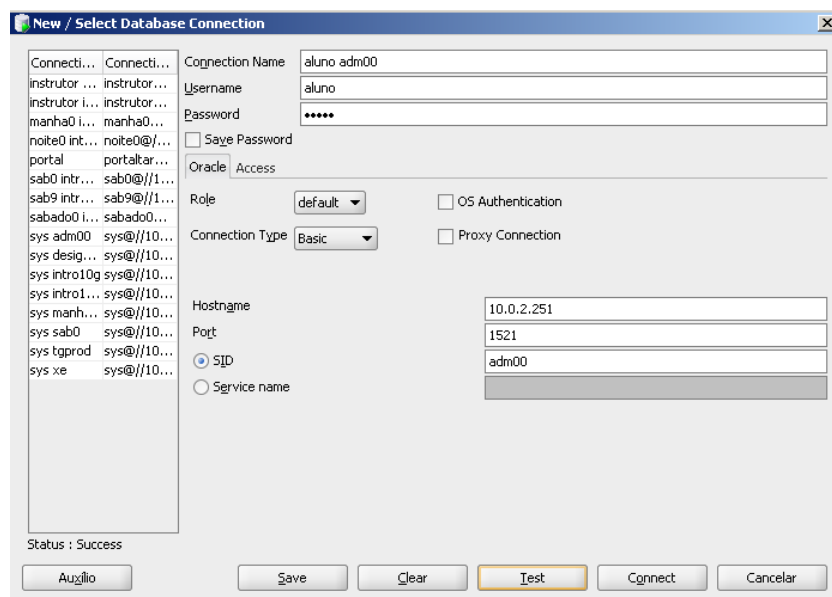
Criando uma conexão no SQL Developer

Selecione a entrada Connections, efetue um clique com o botão direito do mouse e selecione New Connection.

Exemplo: Connection Type TNS



Exemplo: Connection Type Basic



Propriedades da conexão no SQL Developer

Propriedade	Descrição
Connection Name	Nome de identificação da sua conexão
Username	Usuário do banco de dados
Password	Senha do usuário do banco de dados
Role	Default = Conexão normal SYSDBA = Conexão utilizando privilégio especial SYSDBA
Connection Type	TNS = utilizando definições do arquivo TNSNAMES.ORA BASIC = Definições da conexão diretamente no SQL Developer sem utilizar o TNSNAMES.ORA
Network Alias (Para connection type = TNS)	STRING de Conexão com o banco de dados definido no arquivo TNSNAMES.ORA
Hostname	Nome do servidor ou endereço IP do servidor
Port	Porta utilizada pelo Listener
SID	ID da instancia do banco de dados
Service Name	Nome do Serviço

Utilizando o SQL Developer

Executando um comando SQL no SQL Developer

1. Digite o comando SQL
2. Posicione o cursor no comando ou selecione o comando
3. Efetue um clique no botão Execute Statement da toolbar ou a tecla F9.

Executando um Script de comandos SQL e SQL*PLUS no SQL Developer

1. Digite o script comandos SQL e SQL*PLUS
2. Posicione o cursor na área de trabalho comando ou selecione o script a ser executado
3. Efetue um clique no botão Run Script da toolbar ou a tecla F5.

Utilizando o Histórico de Comandos no SQL Developer

1. Efetue um clique no botão SQL History ou a tecla F5.
2. Selecione o comando do histórico.
3. Selecione a opção desejada: APPEND ou REPLACE.
4. APPEND adiciona o comando selecionado na área de trabalho. REPLACE substitue o conteúdo da área de trabalho pelo comando selecionado.

Voltando e Avançando Comandos no SQL Developer utilizando teclas

1. Pressione as Teclas CTRL + Shift + Down para voltar comandos anteriores.
2. Pressione as Teclas CTRL + Shift + Up para avançar comandos.

Exibindo a Estrutura de Tabelas no SQL*PLUS

No SQL*Plus, você pode exibir a estrutura de uma tabela utilizando o comando DESCRIBE. O resultado do comando é uma lista com os nomes de colunas e seus tipos de dados, bem como se a coluna é obrigatória ou não.

```
SQL> DESC[RIBE] nomeobjeto
```

Na sintaxe:

nomeobjeto é o nome de um objeto existente: tabela, visão, sinônimo, procedures e functions.

O exemplo abaixo exibe informações sobre a estrutura da tabela TCLIENTES.

SQL> desc tclientes		
Nome	Nulo?	Tipo
ID	NOT NULL	NUMBER(6)
NOME	NOT NULL	VARCHAR2(35)
DT_NASCIMENTO		DATE
ENDERECO		VARCHAR2(40)
CIDADE		VARCHAR2(30)
ESTADO		VARCHAR2(2)
CEP		VARCHAR2(8)
TELEFONE		VARCHAR2(10)
COMENTARIOS		VARCHAR2(1000)

No resultado:

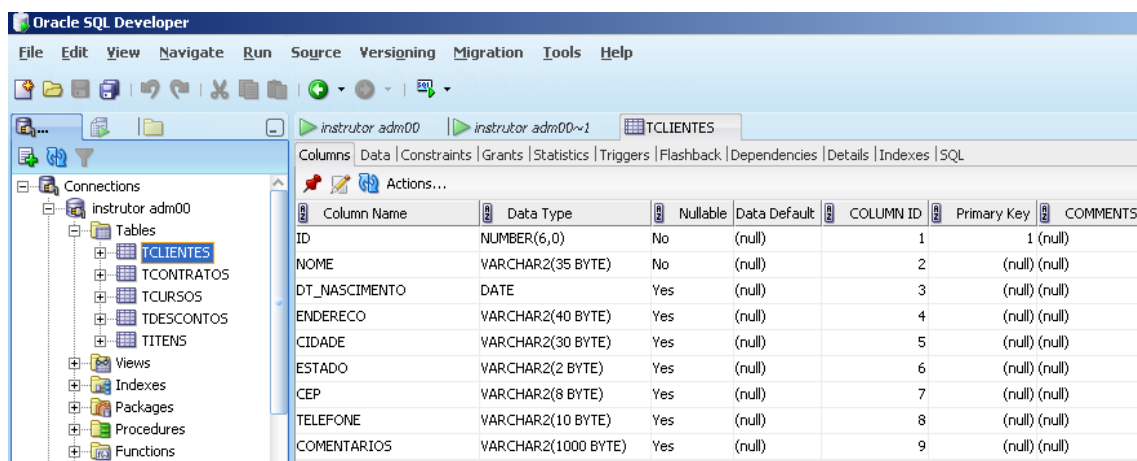
Nulo? - indica quando uma coluna deve conter dados; NOT NULL indica que a coluna é obrigatória.

Tipo - mostra o tipo de dado da coluna.

Exibindo a Estrutura de Tabelas no SQL Developer

No SQL Developer, você pode exibir a estrutura de uma tabela utilizando a ferramenta gráfica.

Selecione a entrada Tables -> Selecione a tabela



Tipos de Dados

Os tipos de dados estão descritos na tabela seguinte:

Tipo de Dado	Descrição
NUMBER(precisão, decimais) e NUMERIC(precisão, decimais)	Valor numérico de tamanho variável contendo um número de máximo de dígitos (total de dígitos) definido por (precisão), sendo o número de dígitos à direita do ponto decimal (decimais) definido por (decimais). A precisão máxima suportada é 38. Se nem a precisão nem os decimais são especificados, então um número com precisão e decimais de até 38 dígitos pode ser utilizado (significa que você pode utilizar um número de até 38 dígitos e podem estar a direita ou a esquerda do ponto decimal).
VARCHAR2(tamanho)	Valor caractere com tamanho variável e até o tamanho máximo definido por (tamanho). Tamanho máximo 4000 bytes.
CHAR(tamanho)	Valor caractere com tamanho fixo de tamanho definido por (tamanho) com preenchimento do final com espaços. Tamanho máximo 2000 bytes.
DATE	Data e hora (dia, mês, ano, hora, minuto e segundo) incluindo o século entre 1 de Janeiro de 4712 A.C. e 31 de dezembro de 9999 D.C. O formato default de apresentação de data é definido pelo parâmetro NLS_DATE_FORMAT (por exemplo: DD/MM/YY).
BINARY_FLOAT	Introduzido no Oracle Database 10g, armazena um numérico de precisão simples (32-bits ponto flutuante). Operações envolvendo BINARY_FLOAT são tipicamente executadas mais rapidamente que operações utilizando NUMBER. BINARY_FLOAT requer 5 bytes de espaço de armazenamento.
BINARY_DOUBLE	Introduzido no Oracle Database 10g, armazena um numérico de precisão dupla (64-bits ponto flutuante). Operações envolvendo BINARY_DOUBLE são tipicamente executadas mais rapidamente que operações utilizando NUMBER. BINARY_DOUBLE requer 9 bytes de espaço de armazenamento.
DEC e DECIMAL	Subtipo de NUMBER. Um número decimal de até 38 dígitos.
REAL	Subtipo de NUMBER. Um número ponto flutuante com até 18 dígitos de precisão.
INT, INTEGER, e SMALLINT	Subtipo de NUMBER. Um inteiro com até 38 dígitos de precisão.
NVARCHAR2(tamanho)	Valor caractere Unicode com tamanho variável e até o tamanho máximo definido por (tamanho). O número de bytes armazenados é 2 multiplicado por (tamanho) para codificação AL16UTF16 e 3 multiplicado por (tamanho) para codificação UTF8. Tamanho máximo 4000 bytes.
NCHAR(tamanho)	Valor caractere Unicode com tamanho fixo de tamanho definido por (tamanho) com preenchimento do final com espaços. O número de bytes armazenados é 2 multiplicado por (tamanho) para codificação AL16UTF16 e 3 multiplicado por (tamanho) para codificação UTF8. Tamanho máximo 2000 bytes.

-
-

Principais Comandos de Arquivo do SQL*Plus

Comandos SQL comunicam com o Servidor Oracle. Comandos SQL*Plus controlam o ambiente, formatam o resultado das consultas e gerenciam arquivos. Você pode utilizar os comandos identificados na tabela seguinte:

Comando	Descrição
SAV[E] filename[.ext] [REPLACE APPEND]	Salva o conteúdo atual do SQL buffer para um arquivo. Utilize a cláusula APPEND para adicionar ao arquivo existente; utilize a cláusula REPLACE para sobrescrever o arquivo existente. A extensão default é .sql.
GET filename[.ext]	Escreve o conteúdo de um arquivo previamente salvo para o SQL buffer. A extensão default para o arquivo é .sql.
STA[RT] filename[.ext]	Executa um arquivo de comandos previamente salvo.
@filename	Executa um arquivo de comandos previamente salvo.
L[ist]	Lista o conteúdo do SQL Buffer.
ED[IT]	Invoca o editor e salva o conteúdo do buffer para um arquivo chamado afiedt.buf.
ED[IT] [filename[.ext]]	Invoca o editor para editar o conteúdo de um arquivo salvo.
SPO[OL] [filename[.ext] OFF OUT APPEND]	Armazena o resultado de consultas em um arquivo. OFF fecha o arquivo de spool. OUT fecha o arquivo de spool enviando-o para a impressora do sistema. APPEND adiciona conteúdo ao arquivo.
EXIT QUIT	Encerra o SQL*Plus

Exercícios – 2

1. Inicie a sessão do SQL Developer criada neste capítulo.
2. Verifique a existência das tabelas utilizadas no curso.
3. Verifique a estrutura da tabela TCLIENTES e depois selecione todas as colunas desta tabela.
4. Faça uma query que selecione as colunas nome, preço e desconto da tabela TCURSOS. Coloque o título da coluna DESCONTO como "Desconto Promocional".
5. Faça um SQL que retorne quais estados possuem clientes.
6. Abra uma sessão do SQL*Plus e mostre a estrutura da tabela TITENS.
7. Existem três erros de codificação neste comando. Você pode os identificar?

```
SELECT id, cod_trg, preço x 0.5 DESCONTO CURSO  
FROM tcursos;
```

8. Crie uma consulta para exibir o id, data de compra, desconto e o total para cada contrato, mostrando o id do contrato por primeiro.

Espaço para anotações

3. Restringindo e Ordenando Dados

Objetivos

- Limitar as linhas recuperadas por uma consulta;
- Ordenar as linhas recuperadas por uma consulta.

Limitando as Linhas Selecionadas

Você pode restringir as linhas recuperadas pela consulta utilizando a cláusula WHERE. A cláusula WHERE contém uma condição que deve ser satisfeita, devendo estar imediatamente após a cláusula FROM.

```
SELECT * | { [DISTINCT] coluna | expressão [alias] , ... }  
FROM tabela  
[WHERE condition(s) ] ;
```

Na sintaxe:

WHERE restringe a consulta para as linhas que satisfazem a condição.

Condition(s) é composta de nomes de coluna, expressões, constantes, e operadores de comparação.

A cláusula WHERE pode comparar valores em colunas, valores literais, expressões aritméticas ou funções. A cláusula WHERE consiste de três elementos:

- Nome da coluna
- Operador de comparação
- Nome de coluna, constante ou lista de valores

Utilizando a Cláusula WHERE

```
SELECT id, nome, dt_nascimento  
FROM tclientes  
WHERE estado = 'RS';
```

ID	NOME	DT_NASCIMENTO
150	Alfredo Fonseca	24/04/70
170	José Batista	05/02/69

No exemplo, o comando SELECT recupera o id, nome e a data de nascimento dos clientes cujo estado é RS.

Observe que o estado RS foi especificado em maiúsculas para assegurar que a comparação feita com a coluna estado da tabela TCLIENTES esteja de acordo com os dados nela armazenados. Comparação de Strings de caractere fazem distinção entre letras maiúsculas e minúsculas.



Strings de Caractere e Datas

Strings de caractere e datas na cláusula WHERE devem ser inseridas entre aspas simples ('). Constantes numéricas, entretanto, não devem estar entre aspas.

Comparações com tipo caractere fazem distinção entre maiúsculas e minúsculas. No exemplo a seguir, nenhuma linha é retornada porque a tabela TCLIENTES armazena todos os dados em maiúsculas.

Exemplo:

```
SELECT id, nome
  FROM tclientes
 WHERE estado = 'rs';
```

 ID	 NOME

O Oracle armazena datas em um formato numérico interno, representando o século, ano, mês, dia, horas, minutos e segundos. A exibição default de datas é configurável.

Valores numéricos não devem ser incluídos dentro de aspas.

Nota: o formato default de exibição de data será visto no capítulo 4.

Operadores de Comparação

Operador	Significado
=	Igual a
>	Maior que
>=	Maior que ou igual a
<	Menor que
<=	Menor que ou igual a
<> ou !=	Diferente de

Figura 3-1: Operadores de comparação

Operadores de comparação são utilizados em condições que comparam uma expressão com outra. Eles são usados na cláusula WHERE no seguinte formato:

Sintaxe

```
... WHERE expr operador valor
```

Exemplos

```
... WHERE dt_nascimento = '01/01/99'  
... WHERE tclientes_id >= 15  
... WHERE nome = 'Amarildo Fagundes'
```

Utilizando os Operadores de Comparação

```
SELECT id, dt_compra, total, desconto,  
       NVL(desconto, 0) + 1000  
FROM tcontratos  
WHERE total <= NVL(desconto, 0) + 1000;
```

ID	DT_COMPRA	TOTAL	DESCONTO	NVL(DESCONTO,0)+1000
1007	06/01/05	1000	50	1050
1012	08/01/05	600	60	1060
1014	08/01/05	956	(null)	1000
1019	11/01/05	600	60	1060

No exemplo, o comando SELECT recupera o id, data de compra, total e o desconto menos 1000 a partir da tabela TCONTRATOS, quando o total for menor que NVL(desconto, 0) + 1000. Os dois valores são comparados e são obtidos a partir das colunas TOTAL e DESCONTO da tabela TCONTRATOS.

Outros Operadores de Comparação

Operador	Significado
BETWEEN ...AND...	Entre dois valores (inclusive)
IN(list)	Igual a um dos valores da lista
LIKE	Igual a um padrão de caracteres
IS NULL	Possui um valor nulo

Figura 3-2: Outros operadores de comparação

Operador BETWEEN

```
SELECT nome, preco
FROM tcursos
WHERE preco BETWEEN 800 and 1100;
```

NOME	PREÇO
Oracle 10G Developer: Forms - Parte II Avançado	956
SQL Standard	1000
SQL Standard Avançado	1000
Otimização de Aplicativos Oracle	956
Modelagem de Dados e Projeto de Banco de Dados	800
Oracle 10G Designer: Modelagem de Sistemas	956
Oracle 10G Designer: Design e Geração do Banco de Dados	956
Oracle 10G Designer: Design e Geração de Reports	956
Oracle 10G Designer: Design e Geração de Aplicações para Web	956
Desenvolvimento de Aplicações Web em PL/SQL	956

Você pode exibir linhas baseadas em um intervalo de valores utilizando o operador BETWEEN. O intervalo que você especifica é composto por um limite inferior e um limite superior.

O comando SELECT acima recupera linhas da tabela TCURSOS para qualquer curso cujo preço está entre R\$800 e R\$1100.

Os valores especificados no operador BETWEEN fazem parte do intervalo, sendo também recuperados. Você deve especificar o limite inferior por primeiro.

Operador IN

Utilize o operador IN para executar a comparação com os valores de uma lista.

```
SELECT nome, dt_nascimento, cidade, estado
FROM tclientes
WHERE estado IN ('RS', 'RJ', 'SP', 'PR');
```

ID	NOME	ID	DT_NASCIMENTO	ID	CIDADE	ID	ESTADO
	Amarildo Fagundes		22/12/76		São Paulo		SP
	Ana Maria Prado		03/03/71		Rio de Janeiro		RJ
	Antônio da Silva		10/07/64		São Paulo		SP
	Ramiro Antunes		19/10/61		Rio de Janeiro		RJ
	Nadia Velasques		20/01/73		Rio de Janeiro		RJ
	Alfredo Fonseca		24/04/70		Porto Alegre		RS
	Jânio Marques		11/05/65		São Paulo		SP
	José Batista		05/02/69		Porto Alegre		RS
	Carlos Magno		30/10/71		São Paulo		SP
	João Medeiros		15/09/65		Rio de Janeiro		RJ
	Mário Cardoso		07/08/77		São Paulo		SP

O exemplo acima exibe id, nome e cidade dos clientes pertencentes aos estados 'RS', 'RJ', 'SP' e 'PR'.

O operador IN pode ser utilizado com qualquer tipo de dado. O exemplo a seguir recupera uma linha a partir da tabela TCLIENTES para cada clientes com seu id listado na cláusula WHERE.

```
SELECT nome, dt_nascimento, cidade, estado
FROM tclientes
WHERE id IN (100, 120, 150);
```

Nota: Se forem utilizadas strings de caractere ou datas na lista, estas devem estar entre aspas simples (').

Operador LIKE

Às vezes você pode não saber o valor exato a pesquisar. Você pode então selecionar linhas que combinem com um padrão de caracteres utilizando o operador LIKE. Podem ser utilizados dois símbolos para construir a string de procura.

Símbolo	Descrição
%	Representa uma sequência de zero, um ou "n" caracteres quaisquer.
_	Representa um único caractere qualquer.

Tabela 3-1: Símbolos do operador LIKE

Exemplo:

```
SELECT nome
FROM tclientes
WHERE nome LIKE 'A%';
```

A	NOME
	Amarildo Fagundes
	Ana Maria Prado
	Antônio da Silva
	Alfredo Fonseca

O comando SELECT acima recupera o nome dos clientes da tabela TCLIENTES para qualquer nome de cliente que começa com um "A". Observe que os nomes que começam com um "a" minúsculo não serão recuperados.

Exemplo:

```
SELECT nome
FROM tclientes
WHERE (nome LIKE '%A%') or (nome LIKE '%a%');
```

A Z	NOME
	Amarildo Fagundes
	Ana Maria Prado
	Antônio da Silva
	Ramiro Antunes
	Nadia Velasques
	Alfredo Fonseca
	Jânio Marques
	José Batista
	Carlos Magno
	Mário Cardoso

O comando SELECT acima recupera o nome dos clientes da tabela TCLIENTES para qualquer nome de cliente que começa possua a letra "A" ou "a" em qualquer posição.

Combinando Padrões de Caractere

Os símbolos % e _ podem ser utilizados em qualquer combinação com literais de caracteres. O exemplo abaixo exibe os nomes de todos os clientes cujo nome possui a letra "A" como o segundo caractere.

```
SELECT id, nome, cidade
FROM tclientes
WHERE nome LIKE '_a%';
```

ID	A Z	NOME	A Z	CIDADE
130		Ramiro Antunes		Rio de Janeiro
140		Nadia Velasques		Rio de Janeiro
180		Carlos Magno		São Paulo

A Opção ESCAPE

Quando você precisar de uma comparação exata para os caracteres "%" e "_", utilize a opção de ESCAPE. Para exibir os nomes de clientes cujo nome contém "a_B", utilize o seguinte comando SQL:

```
SELECT nome
FROM tclientes
WHERE nome LIKE '%a\_B' ESCAPE '\';
```

A opção de ESCAPE identifica a barra invertida (\) como o caracter de escape. No padrão, o caractere de escape precede o underscore (_). Isto faz com que o Servidor Oracle interprete o underscore literalmente.

Operador IS NULL

```
SELECT nome, pre_requisito
FROM tcursos
WHERE pre_requisito IS NULL;
```

NOME	PRE_REQUISITO
Introdução ao Oracle 10G I: Conceitos básicos do Oracle e SQL*PLUS, SQL e SQL Avançado	(null)
SQL Standard	(null)
Modelagem de Dados e Projeto de Banco de Dados	(null)
Oracle 10G Discover para Administradores	(null)
Oracle 10G Discover para Usuários	(null)
Orientação a Objetos - Fundamentos UML, Análise e Projetos de Sistemas	(null)
Introdução ao Linux	(null)
Apache - Administração de Servidor Web	(null)
Segurança de Redes - Firewall	(null)
Segurança de Redes - Ferramentas e Serviços	(null)

O operador IS NULL testa os valores que são nulos. Um valor nulo é um valor que é indisponível, não atribuído, desconhecido ou inaplicável. Portanto, você não pode testá-los com (=) porque um valor nulo não pode ser igual ou diferente de qualquer valor. O exemplo acima recupera o nome de todos os cursos que não possuem pré-requisitos.

Operadores Lógicos

Operador	Significado
AND	Retorna TRUE se ambas as condições resultarem em TRUE
OR	Retorna TRUE se qualquer das condições retornarem TRUE
NOT	Retorna TRUE se a condição seguinte retornar FALSE

Figura 3-3: Operadores lógicos

Um operador lógico combina o resultado de duas condições para produzir um único resultado ou para inverter o resultado de uma única condição. Três operadores lógicos estão disponíveis em SQL:

- AND
- OR
- NOT

Todos os exemplos mostrados até o momento especificaram somente uma condição na cláusula WHERE. Você pode usar várias condições em uma cláusula WHERE utilizando os operadores AND e OR.

Operador AND

```
SELECT id, dt_compra, total, desconto
FROM tcontratos
WHERE total >= 1000
AND desconto IS NOT NULL;
```

ID	DT_COMPRA	TOTAL	DESCONTO
1000	03/01/05	3600	360
1001	04/01/05	4500	225
1003	04/01/05	1995	985
1005	05/01/05	3191	191
1007	06/01/05	1000	50
1008	06/01/05	6218	700
1011	07/01/05	3000	1500
1013	08/01/05	2600	260
1016	09/01/05	15960	960

No exemplo, ambas as condições devem ser verdadeiras para qualquer registro a ser selecionado. Portanto, um contrato que tem o total \geq R\$1000 e possua desconto será selecionado.

Todas as pesquisas do tipo caractere fazem distinção entre maiúsculas e minúsculas.

Combinações de Resultados com o Operador AND

A tabela abaixo mostra o resultado da combinação de duas expressões com o operador AND:

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Tabela 3-2: Combinações de resultados com o operador AND

Operador OR

```
SELECT id, dt_compra, total
FROM tcontratos
WHERE total >= 1000
      OR desconto IS NOT NULL;
```

ID	DT_COMPRA	TOTAL	DESCONTO
1000	03/01/05	3600	360
1001	04/01/05	4500	225
1002	04/01/05	1800	(null)
1003	04/01/05	1995	985
1004	05/01/05	9000	(null)
1005	05/01/05	3191	191
1006	06/01/05	2000	(null)
1007	06/01/05	1000	50
1008	06/01/05	6218	700
1009	06/01/05	1200	(null)

No exemplo, para um registro ser selecionado basta que uma das duas condições seja verdadeira. Portanto, um contrato possui desconto ou tem valor total maior ou igual a R\$1000 será selecionado.

Combinações de Resultados com o Operador OR

A tabela abaixo mostra o resultado da combinação de duas expressões com o operador OR:

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Tabela 3-3: Combinações de resultados com o operador OR

Operador NOT

```
SELECT nome, dt_nascimento, cidade
FROM tclientes
WHERE cidade NOT IN ('São Paulo', 'Rio de Janeiro');
```

NOME	DT_NASCIMENTO	CIDADE
Alfredo Fonseca	24/04/70	Porto Alegre
José Batista	05/02/69	Porto Alegre

O exemplo acima exibe o nome, a data de nascimento e a cidade de todos os clientes cuja cidade não seja 'São Paulo' ou 'Rio de Janeiro'.

Combinações de Resultados com o Operador NOT

A tabela abaixo mostra o resultado da aplicação do operador NOT a uma condição:

	TRUE	FALSE	NULL
NOT	FALSE	TRUE	NULL

Tabela 3-4: Combinações de resultados com o operador NOT

```
... WHERE estado NOT IN ('RJ', 'RS')
... WHERE preco NOT BETWEEN 1000 AND 1500
... WHERE nome NOT LIKE '%A%'
... WHERE desconto IS NOT NULL
```

Nota: O operador NOT também pode ser utilizado com outros operadores SQL como BETWEEN, LIKE e NULL.

Regras de Precedência

Ordem de precedência	Descrição
1	Operadores aritméticos
2	Operador de Concatenação
3	Condições de Comparação
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	NOT EQUAL TO
7	NOT condição lógica
8	AND condição lógica
9	OR condição lógica

Figura 3-4: Regras de precedência dos operadores

Nota: Você pode utilizar parênteses para sobrepor a regra de precedência.

Exemplo:

```
SELECT id, cod_trg, preco
FROM tcursos
WHERE preco < 750
   OR preco > 1500
   AND pre_requisito IS NULL;
```

Portanto, lê-se o comando SELECT como segue:

“Selecione a linha se o curso possuir o preço menor que R\$750 ou o preço do curso for maior que R\$1500, mas sem nenhum pré-requisito.”

R2	ID	R2	COD_TRG	R2	PRECO
	1		TO10G1		1800
	20		TD31U		638
	48		TIL		277
	27		TLFAD5		498
	28		TLASLX		498
	29		TLARLN		498
	30		TLSAMB		665
	31		TLAPAC		665
	47		THTML4		600
	40		TJASCR		600

No exemplo, a precedência é:

1. preço < 750
2. preço > 1500
3. pre_requisito IS NULL

4. Operador AND
5. Operador OR

Utilizando Parênteses para Alterar a Prioridade

```
SELECT id, cod_trg, preco
FROM tcursos
WHERE ( preco < 750
      OR  preco > 1500 )
      AND pre_requisito IS NULL;
```

ID	COD_TRG	PRECO
1	TO10G1	1800
20	TD31U	638
48	TIL	277
31	TLAPAC	665
47	THTML4	600
35	TFWRMX	600
36	TFLSMX	600
50	T50-I	332
77	TFRH10	600

No exemplo, a precedência é:

1. Resolver o conteúdo dos parênteses.
 - a. preço < 750
 - b. preço > 1500
- Operador OR
- c.
2. Pré-requisito IS NULL
3. Operador AND

Portanto, lê-se o comando SELECT como segue:

“Selecione a linha se o curso possuir o preço menor que R\$750 ou maior que R\$1500 e não possua pré-requisitos.”

Cláusula ORDER BY

```
SELECT nome, dt_nascimento, cidade, estado
FROM tclientes
ORDER BY nome;
```

NOME	DT_NASCIMENTO	CIDADE	ESTADO
Alfredo Fonseca	24/04/70	Porto Alegre	RS
Amarildo Fagundes	22/12/76	São Paulo	SP
Ana Maria Prado	03/03/71	Rio de Janeiro	RJ
Antônio da Silva	10/07/64	São Paulo	SP
Carlos Magno	30/10/71	São Paulo	SP
Jânio Marques	11/05/65	São Paulo	SP
João Medeiros	15/09/65	Rio de Janeiro	RJ
José Batista	05/02/69	Porto Alegre	RS
Mário Cardoso	07/08/77	São Paulo	SP
Nadia Velasques	20/01/73	Rio de Janeiro	RJ

A ordem das linhas recuperadas no resultado de uma consulta é indefinida. A cláusula ORDER BY pode ser utilizada para ordenar as linhas. Se utilizada, deve aparecer como a última cláusula do comando SELECT. Você pode ordenar por uma expressão ou por um alias de coluna.

Sintaxe:

```
SELECT expr
FROM table
[WHERE condição(s)]
[ORDER BY {coluna | expressão | alias} [ASC | DESC], ...];
```

Onde:

ORDER BY: especifica a ordem na qual as linhas recuperadas serão exibidas.

ASC: ordena as linhas em ordem ascendente. Esta é a ordenação padrão.

DESC: ordena as linhas em ordem descendente.

Se a cláusula ORDER BY não for utilizada, a ordem em que as linhas serão recuperadas é indefinida, e o Servidor Oracle pode não recuperar as linhas duas vezes na mesma ordem para a mesma consulta. Utilize a cláusula ORDER BY para exibir as linhas em uma ordem específica.

Ainda é possível realizar a ordenação começando pelos valores nulos ou terminando pelos valores nulos: NULLS FIRST e NULLS LAST.

Classificando em Ordem Descendente

```
SELECT nome, dt_nascimento, cidade, estado  
FROM tclientes  
ORDER BY nome DESC;
```

RANK	NOME	RANK	DT_NASCIMENTO	RANK	CIDADE	RANK	ESTADO
	Ramiro Antunes		19/10/61		Rio de Janeiro		RJ
	Nadia Velasques		20/01/73		Rio de Janeiro		RJ
	Mário Cardoso		07/08/77		São Paulo		SP
	José Batista		05/02/69		Porto Alegre		RS
	João Medeiros		15/09/65		Rio de Janeiro		RJ
	Jânio Marques		11/05/65		São Paulo		SP
	Carlos Magno		30/10/71		São Paulo		SP
	Antônio da Silva		10/07/64		São Paulo		SP
	Ana Maria Prado		03/03/71		Rio de Janeiro		RJ
	Amarildo Fagundes		22/12/76		São Paulo		SP

Ordenação Padrão dos Dados

A ordenação padrão dos dados é ascendente:

- Valores numéricos são exibidos com o valor mais baixo primeiro, por exemplo: 1–999.
- Valores tipo data são exibidos com o valor mais antigo primeiro, por exemplo: 01/01/92 antes de 01/01/95.
- Valores caractere são exibidos em ordem alfabética, por exemplo: A primeiro e Z por último.
- Valores nulos são exibidos por último para ordenações ascendentes e por primeiro para ordenações descendentes.

Invertendo a Ordem Padrão

Para inverter a ordem na qual as linhas são exibidas, especifique o palavra chave DESC depois do nome da coluna na cláusula ORDER BY. O exemplo acima ordena o resultado pelo nome do cliente em ordem decrescente.

Ordenando pelo Alias de Coluna

```
SELECT id, dt_compra, total VALOR
FROM tcontratos
ORDER BY VALOR;
```

ID	DT_COMPRA	VALOR
1019	11/01/05	600
1012	08/01/05	600
1014	08/01/05	956
1007	06/01/05	1000
1009	06/01/05	1200
1002	04/01/05	1800
1018	10/01/05	1912
1003	04/01/05	1995
1006	06/01/05	2000
1010	07/01/05	2215

Você pode utilizar um alias de coluna na cláusula ORDER BY. O exemplo acima ordena os dados pelo total do contrato (alias VALOR).

Ordenando pela posição numérica da coluna

```
SELECT id, dt_compra, total VALOR
FROM tcontratos
ORDER BY 1;
```

ID	DT_COMPRA	VALOR
1000	03/01/05	3600
1001	04/01/05	4500
1002	04/01/05	1800
1003	04/01/05	1995
1004	05/01/05	9000
1005	05/01/05	3191
1006	06/01/05	2000
1007	06/01/05	1000
1008	06/01/05	6218
1009	06/01/05	1200

Você pode ordenar pela posição numérica da coluna na cláusula SELECT na cláusula ORDER BY. O exemplo acima ordena os dados pelo id do contrato (Primeira coluna).

Ordenando por Múltiplas Colunas

Referenciando o nome das colunas

```
SELECT estado, nome, dt_nascimento
FROM tclientes
ORDER BY estado, nome DESC;
```

Referenciando a posição relativa das colunas

```
SELECT estado, nome, dt_nascimento
FROM tclientes
ORDER BY 1, 2 DESC;
```

Resultado

ESTADO	NOME	DT_NASCIMENTO
RJ	Ramiro Antunes	19/10/61
RJ	Nadia Velasques	20/01/73
RJ	João Medeiros	15/09/65
RJ	Ana Maria Prado	03/03/71
RS	José Batista	05/02/69
RS	Alfredo Fonseca	24/04/70
SP	Mário Cardoso	07/08/77
SP	Jânio Marques	11/05/65
SP	Carlos Magno	30/10/71
SP	Antônio da Silva	10/07/64

Você pode ordenar os resultados das consultas por mais de uma coluna.

Na cláusula **ORDER BY**, especifique as colunas separando-as por vírgulas. Se você quiser inverter a ordem de uma coluna, especifique **DESC** depois de seu nome. Você pode ordenar por colunas que não estão incluídas na lista da cláusula **SELECT**.

No exemplo, ordenamos todos os clientes da tabela **TCLIENTES** pelo estado em ordem crescente e depois pelo nome em ordem decrescente (primeira coluna que aparece na cláusula **SELECT**).

Exercícios – 3

1. Inicie uma sessão do SQL Developer.
2. Crie uma consulta para exibir o id e a data de compra (dt_compra) dos contratos com o total superior a 10000.
3. Crie uma consulta para exibir o nome, endereço, cidade, cep e telefone do cliente com o id 140.
4. Crie uma consulta para exibir o id, dt_compra, desconto e total dos contratos (tabela tcontratos) para todos os contratos cuja total não está na faixa de 2000 à 5000.
5. Altere a consulta para selecionar somente os contratos que possuem desconto maior do que zero. Apresente o resultado em ordem crescente de data de compra.
6. Mostre o nome, a cidade e o estado de todos os clientes que possuem estado igual a 'RS' ou 'SP' em ordem alfabética.
7. Crie uma consulta para exibir os contratos sem desconto.

Espaço para anotações

4. Funções Single Row, Funções de Conversão e Expressões de Condição

Objetivos

- Conhecer os vários tipos de funções single row disponíveis em SQL;
- Utilizar funções do tipo caractere, numéricas e de datas em comandos SELECT;
- Conhecer o uso de funções de conversão;
- Conhecer as expressões de condição.

Funções SQL Single Row

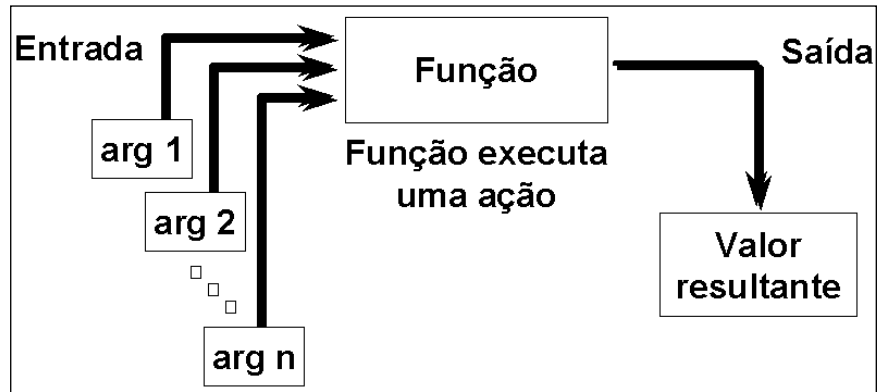


Figura 4-1: Funções SQL

Funções Single Row retorna um valor para cada linha.

Funções são uma característica bastante útil do SQL e podem ser utilizadas para fazer o seguinte:

- Executar cálculos em dados
- Modifique itens de dados individuais
- Manipular um resultado para grupos de linhas
- Formatar datas e números para exibição
- Converter o tipo de dado de colunas

Funções SQL recebem argumento(s) e retornam valor(es).

Tipos de Funções SQL

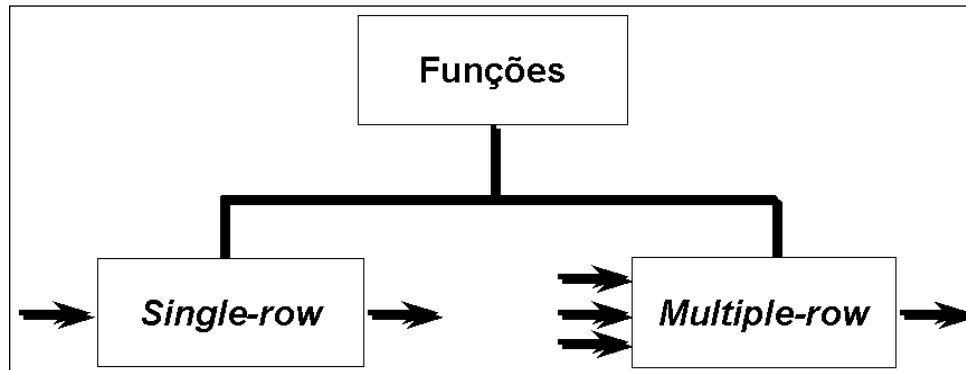


Figura 4-2: Tipos de funções SQL

Existem dois tipos distintos de funções:

- Funções do tipo *single-row*
- Funções do tipo *multiple-row*

Funções do Tipo Single-Row

Estas funções operam em linhas únicas retornando um resultado para cada linha processada. Existem diferentes tipos de funções *single-row*. Este capítulo explica os tipos listados abaixo:

- Caractere
- Numérica
- Data
- Conversão

Funções do Tipo Multiple-Row

Estas funções manipulam grupos de linhas para obter um resultado para cada grupo processado e serão vistas em um capítulo posterior.

Funções do Tipo Single-Row

Funções do tipo single-row são utilizadas para manipular itens de dados. Elas recebem um ou mais argumentos e retornam um único valor para cada linha recuperada pela consulta. Um argumento pode ser um dos seguintes:

- Uma constante fornecida pelo usuário
- Um valor variável
- Um nome de coluna
- Uma expressão

`function_name (coluna | expressão, [arg1, arg2,...])`

Características de Funções Tipo Single-Row

- Atuam sobre cada linha recuperada pela consulta.
- Retornam um resultado por linha.
- Podem retornar um valor com o tipo de dado diferente do referenciado.
- Podem receber um ou mais argumentos.

Você pode utilizá-las nas cláusulas SELECT, WHERE, HAVING e ORDER BY. Você pode também aninhar funções.

Na sintaxe:

function_name: é o nome da função.

column: é qualquer coluna nomeada do banco de dados.

expression: é qualquer string de caractere ou expressão calculada.

arg1, arg2: são quaisquer argumentos a serem utilizadas pela função.

Funções single-row

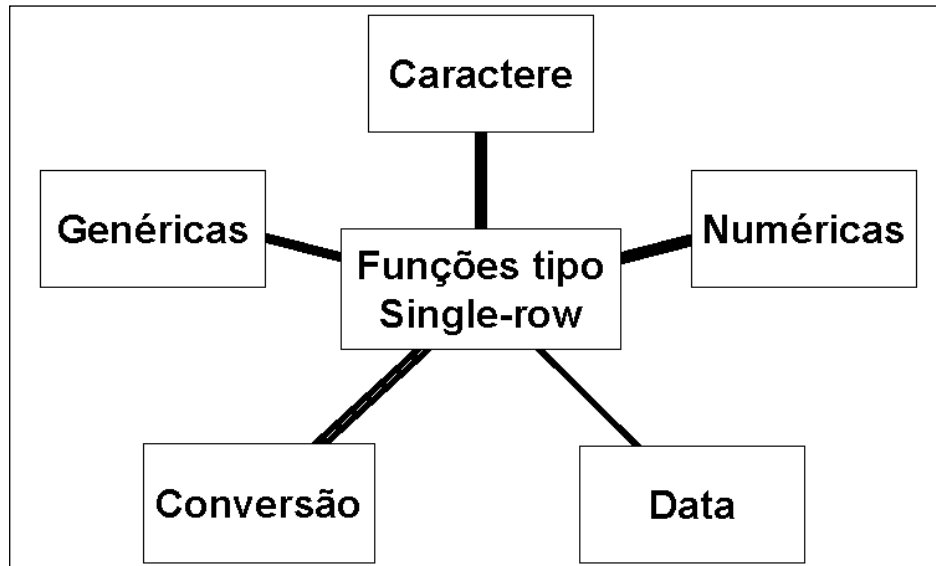


Figura 4-3: Funções single-row

Este capítulo explica as seguintes funções do tipo *single-row*:

- Funções de caracteres: Recebem parâmetros caractere e podem retornar valores numéricos ou caractere.
- Funções numéricas: Recebem parâmetros numéricos e retornam valores numéricos.
- Funções de data: Operam em valores do tipo de dado DATE. Todas as funções de data retornam um valor do tipo de dado data, exceto a função MONTHS_BETWEEN que retorna um número.
- Funções de conversão: Convertem um valor de um tipo de dado para outro.
- Funções genéricas:
 - NVL
 - NVL2
 - DECODE

Funções de Caracteres

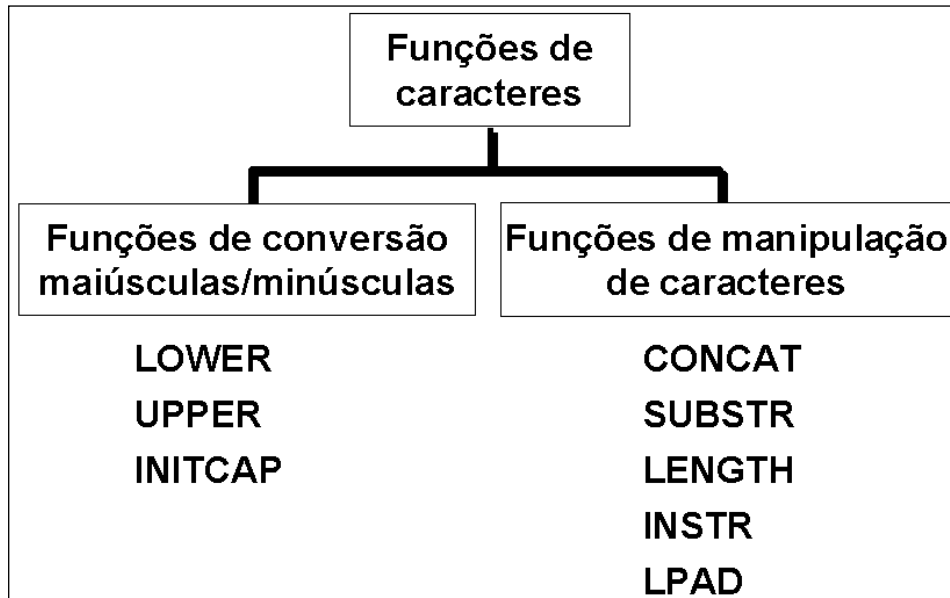


Figura 4-4: Funções de caracteres

Funções de caracteres aceitam dados do tipo caractere como entrada e podem retornar valores do tipo caractere ou numérico. Funções de caracteres podem ser divididas em:

- Funções de conversão entre maiúsculas/minúsculas
- Funções de manipulação de caracteres

Funções de Caracteres (continuação)

Função	Propósito
CONCAT (<i>column1</i> <i>expression1</i> , <i>column2</i> <i>expression2</i>)	Concatena a primeira string de caracteres com a segunda. Equivalente ao operador de concatenação ()
INITCAP (<i>column</i> <i>expression</i>)	Converte strings de caracteres deixando a primeira letra de cada palavra em maiúscula e as demais em minúsculas
INSTR (<i>column</i> <i>expression</i> , <i>m</i>)	Retorna a posição numérica do caracter dentro da string
LENGTH (<i>column</i> <i>expression</i>)	Retorna o número de caracteres da string
LOWER (<i>column</i> <i>expression</i>)	Converte strings de caracteres para minúsculas
LPAD (<i>column</i> <i>expression</i> , <i>n</i> , ' <i>string</i> ')	Retorna uma string com tamanho total de <i>n</i> alinhada à direita
REPLACE (<i>x</i> , <i>search_string</i> , <i>replace_string</i>)	Procura a string <i>search_string</i> em <i>x</i> e substitui por <i>replace_string</i>
RPAD (<i>column</i> <i>expression</i> , <i>n</i> , ' <i>string</i> ')	Retorna uma string com tamanho total de <i>n</i> alinhada à esquerda
SUBSTR(<i>column</i> <i>expression</i> , <i>m</i> , <i>n</i>)	Retorna os caracteres especificados a partir da string de caracteres, começando na posição <i>m</i> , com tamanho de <i>n</i> caracteres. Se <i>m</i> for negativo, a contagem inicia a partir do final da string. Se <i>n</i> for omitido, são retornados todos os caracteres até o final da string
UPPER(<i>column</i> <i>expression</i>)	Converte strings de caracteres para maiúsculas.
TRIM([<i>trim_char</i> FROM] <i>x</i>)	Remove caracteres da esquerda e direita do string <i>x</i> . Você pode fornecer um caracter opcional que especifica o caracter (<i>trim_char</i>) a ser removido; se você não fornecer, espaços serão removidos por default.
LTRIM(<i>x</i> [, <i>trim_char</i>])	Remove caracteres da esquerda do string <i>x</i> . Você pode fornecer um caracter opcional que especifica o caracter (<i>trim_char</i>) a ser removido; se você não fornecer, espaços serão removidos por default.
RTRIM(<i>x</i> [, <i>trim_char</i>])	Remove caracteres da direita do string <i>x</i> . Você pode fornecer um caracter opcional que especifica o caracter (<i>trim_char</i>) a ser removido; se você não fornecer, espaços serão removidos por default.

Tabela 4-1: Funções de caracteres

Funções de Conversão entre Maiúsculas/Minúsculas

Função	Resultado
LOWER('Introdução ORACLE 10g')	introdução oracle 10g
UPPER('Introdução ORACLE 10g')	INTRODUÇÃO ORACLE 10G
INITCAP('Introdução ORACLE 10g')	Introdução Oracle 10g

Tabela 4-2: Funções de conversão entre maiúsculas/minúsculas

LOWER, UPPER e INITCAP são as três funções de conversão entre maiúsculas e minúsculas.

- LOWER: converte todos os caracteres de uma string para minúsculas
- UPPER: converte todos os caracteres de uma string para maiúsculas
- INITCAP: converte a primeira letra de cada palavra para maiúsculas e as demais para minúsculas

```
SELECT 'O Cliente ' || INITCAP(nome) || ' mora em ' ||  
INITCAP(cidade) || '-' || UPPER(estado) AS "Informações"  
FROM tclientes;
```

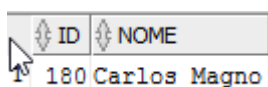
Informações
O Cliente Amarildo Fagundes mora em São Paulo-SP
O Cliente Ana Maria Prado mora em Rio De Janeiro-RJ
O Cliente Antônio Da Silva mora em São Paulo-SP
O Cliente Ramiro Antunes mora em Rio De Janeiro-RJ
O Cliente Nadia Velasques mora em Rio De Janeiro-RJ
O Cliente Alfredo Fonseca mora em Porto Alegre-RS
O Cliente Jânio Marques mora em São Paulo-SP
O Cliente José Batista mora em Porto Alegre-RS
O Cliente Carlos Magno mora em São Paulo-SP
O Cliente João Medeiros mora em Rio De Janeiro-RJ

Utilizando Funções de Conversão entre Maiúsculas e Minúsculas

```
SELECT id, nome  
FROM tclientes  
WHERE nome = 'CARLOS MAGNO';
```

O exemplo acima não retorna nenhuma linha porque não existe nenhum cliente com o nome = 'CARLOS MAGNO' em letras maiúsculas.

```
SELECT id, nome  
FROM tclientes  
WHERE LOWER(nome) = 'carlos magno';
```



ID	NOME
180	Carlos Magno

A cláusula WHERE do primeiro comando SQL especifica o nome do cliente como 'CARLOS MAGNO'. Uma vez que os dados na tabela TCLIENTES não estão armazenados em maiúsculas, o nome 'CARLOS MAGNO' não possui um correspondente na tabela TCLIENTES e como resultado nenhuma linha é selecionada.

A cláusula WHERE do segundo comando SQL especifica que o nome do cliente na tabela TCLIENTES deve ser convertido para minúsculas e então comparado com 'carlos magno'. Considerando que ambos os nomes estão em minúsculas agora, uma correspondência é encontrada e uma linha é selecionada. A cláusula WHERE pode ser reescrita da seguinte maneira para obter o mesmo resultado:

O nome no resultado da consulta aparece como foi armazenado no banco de dados. Para exibir o nome somente com a primeira letra em maiúscula, utilize a função INITCAP no comando SELECT.

Funções de Manipulação de Caracteres

Função	Resultado
CONCAT('Introdução ORACLE 12c',' Curso 1')	Introdução ORACLE 12c Curso 1
SUBSTR('Introdução ORACLE 12c',1,11)	Introdução
LENGTH('Introdução ORACLE 12c')	21
INSTR('Introdução ORACLE 12c','ORACLE')	12
LPAD('Introdução ORACLE 12c',30,'*')	*****Introdução ORACLE 12c
RPAD('Introdução ORACLE 12c',30,'*')	Introdução ORACLE 12c*****
REPLACE('Introdução ORACLE 11g','11g','12c')	Introdução ORACLE 12c
TRIM('; ' FROM 'nome@gmail.com;')	nome@gmail.com
RTRIM('nome@gmail.com;', ' ;')	nome@gmail.com
LTRIM(' nome@gmail.com', ' ')	nome@gmail.com

Tabela 4-3: Funções de manipulação de caracteres

Utilizando as Funções de Manipulação de Caracteres

```
SELECT nome, CONCAT('UF-', estado), LENGTH(nome), INSTR(nome, 'A')
FROM tclientes
WHERE SUBSTR(cidade,1,5) = 'Porto';
```

NOME	CONCAT('UF-',ESTADO)	LENGTH(NOME)	INSTR(NOME,'A')
Alfredo Fonseca	UF-RS	15	1
José Batista	UF-RS	12	0

O exemplo acima exibe o nome do cliente, a string 'UF-' concatenado com o estado, o tamanho do nome do cliente e a posição numérica da letra A no nome, para todos os clientes que possuem a string 'Porto' nas 5 primeiras posições do campo CIDADE da tabela TCLIENTES.

Funções Numéricas

- **ROUND:** **Arredonda o valor para a decimal especificada**
 ROUND(45.926, 2) ➡ 45.93
- **TRUNC:** **Trunca o valor para a decimal especificada**
 TRUNC(45.926, 2) ➡ 45.92
- **MOD:** **Retorna o resto da divisão**
 MOD(1600, 300) ➡ 100

Figura 4-5: Funções numéricas

Funções numéricas recebem parâmetros numéricos e retornam valores numéricos. Esta seção descreve algumas das funções numéricas.

Função	Propósito
ROUND(<i>column expression, n</i>)	Arredonda a coluna, expressão ou valor para <i>n</i> casas decimais. Se <i>n</i> for omitido, será considerado como 0, ou seja, sem casas decimais. Se <i>n</i> for negativo, os números à esquerda do ponto decimal serão arredondados.
TRUNC(<i>column expression, n</i>)	Trunca a coluna, expressão ou valor para <i>n</i> casas decimais. Se <i>n</i> for omitido, será considerado como 0. Se <i>n</i> for negativo, os números à esquerda do ponto decimal serão truncados para zero.
MOD(<i>m, n</i>)	Retorna o resto da divisão de <i>m</i> por <i>n</i> .
ABS(<i>n</i>)	Retorna o valor absoluto de <i>n</i> .
SQRT(<i>n</i>)	Retorna a raiz quadrada de <i>n</i> .

Tabela 4-4: Funções numéricas

Utilizando a Função ROUND

A função ROUND arredonda uma coluna, expressão ou valor para n casas decimais.

```
SELECT ROUND (45.923,2) , ROUND (45.923,0)
FROM DUAL;
```

ROUND(45.923,2)	ROUND(45.923,0)
45,92	46



Se o segundo argumento é 0 ou não for informado, o valor é arredondado para zero casas decimais. Se o segundo argumento é 2, o valor é arredondado para duas casas decimais.

A função ROUND também pode ser utilizada com funções de data.

Utilizando a Função TRUNC

A função TRUNC trunca a coluna, expressão ou valor para n casas decimais.

```
SELECT TRUNC (45.923,2) , TRUNC (45.923)
FROM DUAL;
```

 TRUNC(45.923,2)	 TRUNC(45.923)
45,92	45

A função TRUNC opera com argumentos semelhantes aos da função ROUND. Se o segundo argumento é 0 ou não for informado, o valor é truncado para zero casas decimais. Se o segundo argumento é 2, o valor é truncado para duas casas decimais.

Como a função ROUND, a função TRUNC também pode ser utilizada com funções de data.

Utilizando a Função MOD

```
SELECT id,tclientes_id cliente,desconto,total,  
       MOD(total, NVL(desconto,0))  
FROM tcontratos;
```

ID	CLIENTE	DESCONTO	TOTAL	MOD(TOTAL,DESCONTO)
1000	150	360	3600	0
1001	100	225	4500	0
1003	160	985	1995	25
1005	190	191	3191	135
1007	170	50	1000	0
1008	100	700	6218	618
1011	140	1500	3000	0
1012	150	60	600	0
1013	100	260	2600	0
1016	150	960	15960	600

A função MOD encontra o resto da divisão do valor 1 pelo valor 2. O exemplo acima calcula o resto da divisão do total pelo desconto dos contratos que possuam desconto válido.

Trabalhando com Datas

Formato de Data Oracle

O Oracle armazena datas em um formato numérico interno, representando o século, ano, mês, dia, horas, minutos e segundos.

Datas válidas do Oracle estão entre 1 de janeiro de 4712 A.C. e 31 de dezembro de 9999 D.C.

SYSDATE

SYSDATE é uma função de data que retorna a data e hora atual. Você pode utilizar SYSDATE da mesma maneira que você utiliza qualquer outro nome de coluna. Por exemplo, você pode exibir a data atual selecionando SYSDATE a partir de uma tabela. É habitual selecionar SYSDATE a partir de uma tabela dummy chamada DUAL.

DUAL

A tabela DUAL pertence ao usuário SYS e pode ser acessada por todos os usuários. Contém uma coluna, DUMMY, e uma linha com o valor X. A tabela DUAL é útil quando você quer retornar um valor uma única vez, podendo este valor ser uma constante, pseudocoluna ou expressão que não são derivadas de uma tabela com dados de usuário.

Exemplo:

Mostre a data corrente utilizando a tabela DUAL.

```
SELECT SYSDATE  
FROM dual;
```

Formato Padrão de Datas

O formato padrão para exibição e entrada de datas é especificado pelo parâmetro do banco de dados chamado NLS_DATE_FORMAT.

Somente o DBA pode modificar o valor desse parâmetro. Mas você pode modificar o formato padrão na sua sessão do SQL* PLUS utilizando o comando alter session.

Por exemplo, temos o nosso formato padrão de exibição como 'DD/MM/YY', e desejamos modificar para 'DD-MON-YY', executamos os seguintes comandos:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YY';
```

Agora temos nossa sessão no formato modificado.

Cálculos com Datas

Uma vez que o banco de dados armazena as como números, você pode executar cálculos utilizando os operadores aritméticos como a adição e subtração. Você pode adicionar e subtrair constantes numéricas e datas.

Você pode executar as seguintes operações:

Operação	Resultado	Descrição
data + número	Data	Adiciona um número de dias para uma data
data – número	Data	Subtrai um número de dias a partir de uma data
data – data	Número de dias	Subtrai uma data a partir de outra
data + número/24	Data	Adiciona um número de horas para uma data

Tabela 4-5: Cálculos com datas

Utilizando Operadores Aritméticos com Datas

```
SELECT nome, ROUND(((SYSDATE-dt_nascimento) / 7), 2) SEMANAS  
FROM tclientes  
WHERE estado = 'SP';
```

NOME	SEMANAS
Amarildo Fagundes	1691,93
Antônio da Silva	2341,64
Jânio Marques	2298,07
Carlos Magno	1960,5
Mário Cardoso	1659,35

O exemplo acima exibe o nome e o número de semanas de vida dos clientes com o estado igual a 'SP'. Ele subtrai a data atual (SYSDATE) a partir da data na qual o cliente nasceu e divide o resultado por 7 para calcular o número de semanas.

Nota: SYSDATE é uma função SQL que retorna a data e hora atual. Seus resultados podem diferir do exemplo.

Funções de Data

Função	Descrição
MONTHS_BETWEEN	Número de meses entre duas datas
ADD_MONTHS	Adiciona meses do calendário para uma data
NEXT_DAY	Próximo dia da data especificada
LAST_DAY	Último dia do mês
ROUND	Data arredondada
TRUNC	Data truncada

Figura 4-6: Funções de data

Funções de data operam em datas Oracle. Todas as funções de data retornam um valor do tipo de dado DATE, exceto a função MONTHS_BETWEEN que retorna um valor numérico.

- MONTHS_BETWEEN(date1, date2): Encontra o número de meses entre date1 e date2. O resultado pode ser positivo ou negativo. Se date1 é posterior a date2, o resultado é positivo; se date1 é mais recente que date2, o resultado é negativo. A parte não inteira do resultado representa uma porção do mês.
- ADD_MONTHS(date, n): Adiciona n número de meses do calendário para a data. n deve ser um inteiro e pode ser negativo.
- NEXT_DAY(date, 'char'): Encontra a data do próximo dia da semana ('char') a partir da data do parâmetro date; 'char' pode ser um número representando um dia ou uma string de caracteres.
- LAST_DAY(date): Encontra a data do último dia do mês a partir da data especificada no parâmetro date.
- ROUND(date [, 'fmt']): Retorna a data arredondada para a unidade especificada pelo formato fmt. Se o formato fmt for omitido, a data é arredondada para a data mais próxima.
- TRUNC(date [, 'fmt']): Retorna a data com a porção de tempo do dia truncada à unidade especificada pelo formato fmt. Se o formato fmt for omitido, a data é truncada para o dia mais próximo.

Utilizando Funções de Data





Função	Resultado
MONTHS_BETWEEN('01/01/05','01/01/04')	12
ADD_MONTHS('01/01/05',6)	01/07/05
NEXT_DAY('01/01/05','DOMINGO')	02/01/05
LAST_DAY('01/01/05')	31/01/05

Tabela 4-6: Funções de data

Exemplo:

Para todos os clientes que nasceram a menos de 400 meses, selecione o número do cliente, a data de nascimento, o número de meses vividos, a data precisa após 6 meses do nascimento deste cliente, a primeira sexta-feira após a data de nascimento e o último dia do mês do cliente.

```
SELECT MONTHS_BETWEEN('01/01/05','01/01/04'),
       ADD_MONTHS('01/01/05',6),
       NEXT_DAY('01/01/05','DOMINGO'),
       LAST_DAY('01/01/05')
FROM dual;
```

 MONTHS_BETWEEN('01/01/05','01/01/04')	 ADD_MONTHS('01/01/05',6)	 NEXT_DAY('01/01/05','DOMINGO')	 LAST_DAY('01/01/05')
12	01/07/05	02/01/05	31/01/05

Arredondando e truncando datas

Função	Resultado
ROUND(TO_DATE('25/07/05'),'MONTH')	01/08/05
ROUND(TO_DATE('25/07/05'),'YEAR')	01/01/06
TRUNC(TO_DATE('25/07/05'),'MONTH')	01/07/05
TRUNC(TO_DATE('25/07/05'),'YEAR')	01/01/05

Tabela 4-7: Funções de data

Função	Resultado
TRUNC(sysdate,'DAY')	Data do dia com a hora zero
TRUNC(sysdate)	Data do dia com a hora zero

As funções ROUND e TRUNC podem ser utilizadas para valores numéricos e de data. Quando utilizadas com datas, elas arredondam ou truncam para o modelo de formato especificado. Portanto, você pode arredondar datas para o mais próximo ano ou mês.

Exemplo:

Compare as datas de nascimento para todos os clientes que nasceram em 1971. Mostre o número do cliente, a data de nascimento e o mês as funções ROUND e TRUNC.

```
SELECT id, dt_nascimento,
       ROUND(dt_nascimento, 'MONTH'),
       TRUNC(dt_nascimento, 'MONTH')
FROM tclientes
WHERE dt_nascimento LIKE '%71';
```

ID	DT_NASCIMENTO	ROUND(DT_NASCIMENTO,'MONTH')	TRUNC(DT_NASCIMENTO,'MONTH')
110	03/03/71	01/03/71	01/03/71
180	30/10/71	01/11/71	01/10/71

Funções de Conversão

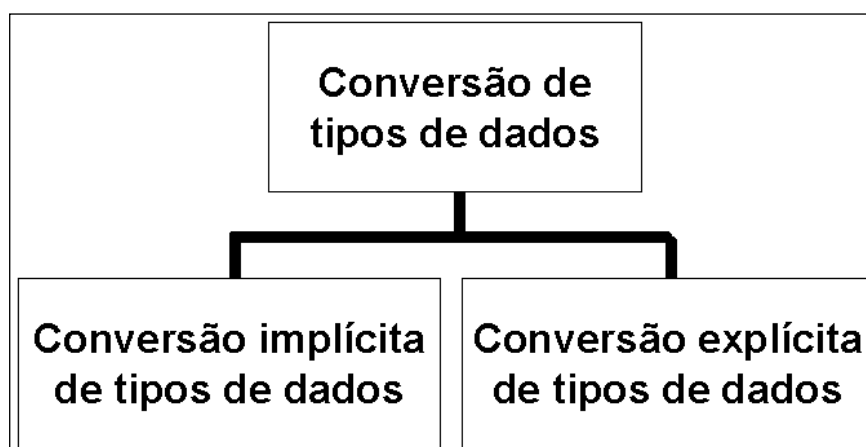


Figura 4-7: Funções de conversão

Além dos tipos de dados Oracle, as colunas de uma tabela no banco de dados Oracle12c podem ser definidas utilizando tipos de dados ANSI, DB2, etc. Entretanto, o Servidor Oracle internamente converte tais tipos de dados para tipos de dados Oracle12c.

Em alguns casos, o Servidor Oracle permite dados de um determinado tipo onde ele esperava dados de um tipo diferente. Isto é permitido porque o Servidor Oracle pode converter os dados automaticamente para o tipo de dado esperado. Esta conversão de tipo de dado pode ser feita implicitamente pelo Servidor Oracle ou explicitamente pelo usuário.

Conversões implícitas de tipo de dado ocorrem de acordo com as regras apresentadas a seguir.

Conversões explícitas de tipo de dado são feitas utilizando as funções de conversão. As funções de conversão convertem um valor de um tipo de dado para outro.

Nota: Embora a conversão de tipo de dado implícita esteja disponível, é recomendado que você faça a conversão explícita para assegurar confiabilidade de seus comandos SQL.

Conversão Implícita de Tipos de Dados

Para atribuições, o Servidor Oracle pode converter automaticamente o seguinte:

- VARCHAR2 ou CHAR para NUMBER
- VARCHAR2 ou CHAR para DATE
- NUMBER para VARCHAR2
- DATE para VARCHAR2

A atribuição tem sucesso se o Servidor Oracle puder converter o tipo de dado do valor utilizado na atribuição para o tipo de dado do destino da atribuição.

Para a avaliação de uma expressão, o Servidor Oracle pode converter automaticamente o seguinte:

- VARCHAR2 ou CHAR para NUMBER

- VARCHAR2 ou CHAR para DATE

Em geral, o Servidor Oracle utiliza a regra para expressões quando uma conversão de tipo de dado é necessária e a regra de conversão em atribuições não pode ser aplicada.

Nota: Conversões de CHAR para NUMBER só tem sucesso se a string de caracteres representa um número válido.

Conversão Explícita de Tipos de Dados

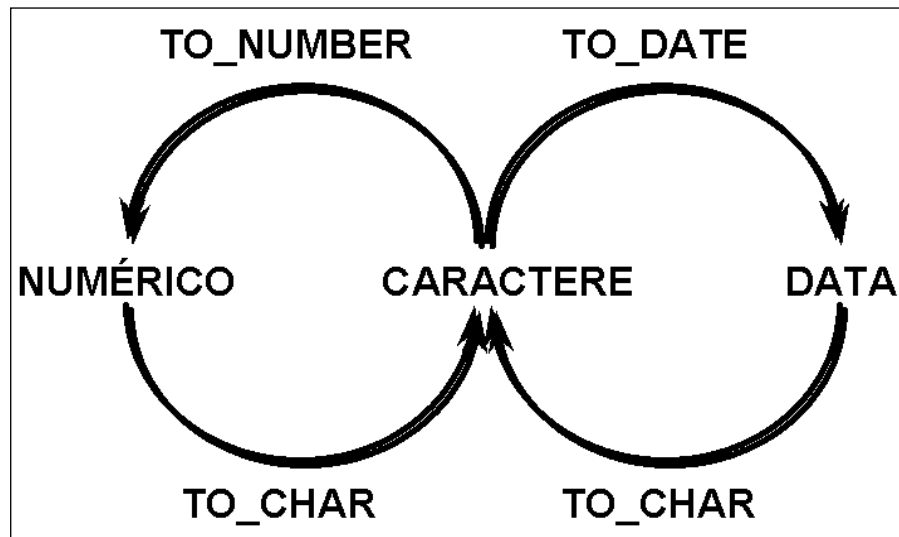


Figura 4-8: Conversão explícita de tipos de dados

O SQL provê três funções para converter um valor de um tipo de dado para outro.

Função	Propósito
TO_CHAR(<i>number</i> <i>date</i> ,[' <i>fmt</i> '])	Converte um valor numérico ou data para uma string de caracteres do tipo VARCHAR2 no formato <i>fmt</i>
TO_NUMBER(<i>char</i>)	Converte uma string de caracteres contendo apenas dígitos para um número
TO_DATE(<i>char</i> ,[' <i>fmt</i> '])	Converte uma string de caracteres representando uma data para um valor de data de acordo com o formato <i>fmt</i> especificado

Tabela 4-8: Conversão explícita de tipos de dados

Função TO_CHAR com Datas

```
TO_CHAR(date, 'fmt')
```

Exibindo uma Data em um Formato Específico

Os valores de data no Oracle são exibidos por default no formato do parâmetro NLS_DATE_FORMAT. A função TO_CHAR permite converter uma data para um formato específico.

Diretrizes:

- A máscara de formatação deve ser incluída entre aspas simples e faz distinção entre maiúsculas e minúsculas.
- A máscara de formatação pode incluir qualquer elemento de formato de data válido.
- Os nomes de dias e meses são automaticamente preenchidos com espaços em branco.
- Para remover espaços em branco ou suprimir zeros à esquerda, utilize o elemento de formatação fm (fill mode).
- Você pode redimensionar a largura de exibição do campo caractere resultante com o comando COLUMN do SQL*Plus.
- A largura da coluna resultante é por default 80 caracteres.

```
SELECT TO_CHAR(sysdate, 'DD/MM/YYYY HH24:MI:SS')  
FROM dual;
```

```
TO_CHAR(SYSDATE,'DD/MM/YYYYHH24:MI:SS')  
26/05/2009 16:51:06
```

Elementos de Formatação de Datas

Elemento	Descrição
CC	Século;
YYYY	Ano com 4 dígitos
YYY ou YY ou Y	Últimos 3, 2 ou 1 dígito(s) do ano
Y,YYY	Ano com símbolo de milhar na posição da ","
BC ou AD	Indicador AC / DC
B.C. ou A.D.	Indicador AC / DC com pontos
Q	Quarto do ano
MM	Valor do mês com dois dígitos
MONTH	Nome do mês com brancos para completar o tamanho de 9 caracteres
MON	Nome do mês abreviado com três letras
RM	Número do mês em Romano
WW ou W	WW = Numero da semana do ano, W = numero da semana do mês
DDD ou DD ou D	Dia do ano, mês ou dia da semana
DAY	Nome do dia com brancos para completar o tamanho de 9 caracteres
DY	Nome do dia abreviado com três letras

Tabela 4-9: Elementos de formatação de data

Formatos de Hora

Utilize os formatos listados na tabela abaixo para exibir informações de hora e literais e para mudar números para números soletrados.

Elemento	Descrição
AM ou PM	Período
HH ou HH12 ou HH24	HH ou HH12 = Hora (1–12) ou HH24 = hora (0–23)
MI	Minuto (0–59)
SS	Segundo (0–59)
SSSSS	Segundos decorridos desde à meia noite (0–86399)

Tabela 4-10: Formatos de hora

Outros Formatos

Elemento	Descrição
/ . , ou espaços	A pontuação é reproduzida no resultado
" de "	Strings entre aspas duplas são reproduzidas no resultado

Tabela 4-11: Outros formatos

Sufixos podem Influenciar a Exibição de Números

Elemento	Descrição
TH	Número ordinal (por exemplo, DDTH para 4TH)
SP	Número soletrado (por exemplo, DDSP para FOUR)
SPTH ou THSP	Número ordinal soletrado (por exemplo, DDSPTH para FOURTH)

Tabela 4-12: Sufixos de exibição de números

Utilizando a Função TO_CHAR com Datas

```
SELECT TO_CHAR(sysdate, 'DD/MM/YYYY HH24:MI:SS "Século" CC')
FROM dual;
```

R	TO_CHAR(SYSDATE,'DD/MM/YYYYHH24:MI:SS"SÉCULO"CC')
26/05/2009 17:07:21	Século 21

```
SELECT nome, TO_CHAR(dt_nascimento, 'DD, "de" month "de" YYYY')
FROM tclientes;
```

R	NOME	R	TO_CHAR(DT_NASCIMENTO,'DD,"DE"MONTH"DE"YYYY')
Amarildo Fagundes	22, de dezembro	de 1976	
Ana Maria Prado	03, de março	de 1971	
Antônio da Silva	10, de julho	de 1964	
Ramiro Antunes	19, de outubro	de 1961	
Nadia Velasques	20, de janeiro	de 1973	
Alfredo Fonseca	24, de abril	de 1970	
Jânio Marques	11, de maio	de 1965	
José Batista	05, de fevereiro	de 1969	
Carlos Magno	30, de outubro	de 1971	
João Medeiros	15, de setembro	de 1965	

```
SELECT nome, TO_CHAR(dt_nascimento, 'FMDD, "de" month "de" YYYY')
TO_CHAR(dt_nascimento, 'FMDD, "de" month "de" YYYY')
FROM tclientes;
```

R	NOME	R	TO_CHAR(DT_NASCIMENTO,'FMDD,"DE"MONTH"DE"YYYY')
Amarildo Fagundes	22, de dezembro	de 1976	
Ana Maria Prado	3, de março	de 1971	
Antônio da Silva	10, de julho	de 1964	
Ramiro Antunes	19, de outubro	de 1961	
Nadia Velasques	20, de janeiro	de 1973	
Alfredo Fonseca	24, de abril	de 1970	
Jânio Marques	11, de maio	de 1965	
José Batista	5, de fevereiro	de 1969	
Carlos Magno	30, de outubro	de 1971	
João Medeiros	15, de setembro	de 1965	

Função TO_CHAR com Números

```
TO_CHAR(number, 'fmt')
```

Quando trabalhar com valores numéricos como se fossem strings de caractere, você deve converter esses números para o tipo de dado caractere utilizando a função TO_CHAR que traduz um valor do tipo de dado NUMBER para o tipo de dado VARCHAR2. Esta técnica é especialmente útil em concatenações.

Elementos de Formatação de Números

Se você está convertendo um número para o tipo de dado caractere, você pode utilizar os elementos listados abaixo:

Elemento	Descrição
9	Posição numérica (o número de 9s determina o tamanho da exibição)
0	Complementa com zeros a partir da posição especificada
\$	Indicador de dólar flutuante
L	Símbolo flutuante da moeda local (Conforme especificação do parâmetro NLS_CURRENCY).
D	Símbolo decimal na posição especificada (conforme especificação do parâmetro NLS_NUMERIC_CHARACTER)
G	Símbolo de milhar na posição especificada (conforme especificação do parâmetro NLS_NUMERIC_CHARACTER)
MI	Sinal de menos à direita (valores negativos)
PR	Números negativos entre parênteses
EEEE	Notação científica (o formato deve especificar quatro Es)

Tabela 4-13: Elementos de formatação de números

Exemplos:

Função	Resultado
TO_CHAR(1234,'999999')	1234
TO_CHAR(1234,'099999')	001234
TO_CHAR(1234,'\$999999')	\$1234
TO_CHAR(1234,'L999999')	R\$1234
TO_CHAR(1234,'9999D99')	1234,00
TO_CHAR(1234,'999G999')	1.234
TO_CHAR(-1234,'999999MI')	1234-
TO_CHAR(-1234,'999999PR')	<1234>
TO_CHAR(1234,'999999EEEE')	1E+03

Tabela 4-14: Exemplos de utilização da função TO_CHAR

Utilizando a Função TO_CHAR com Números

```
SELECT id, dt_compra, to_char(total, 'L99G999G999D99')  
FROM tcontratos;
```

Diretrizes

- O Servidor Oracle exibe uma string com o caractere (#) ao invés de um número quando o número de dígitos exceder o número de dígitos providos pela máscara.
- O Servidor Oracle arredonda o valor decimal armazenado para o número de espaços decimais providos pela máscara.

Funções TO_NUMBER e TO_DATE

`TO_NUMBER(char)`

`TO_DATE(char[, 'fmt'])`

Você pode querer converter uma string de caracteres para um número ou uma data. Para realizar esta tarefa, você utiliza as funções TO_NUMBER e TO_DATE. A máscara que você pode escolher baseia-se nos elementos de formato apresentados anteriormente.

Exemplo:

Mostre os nomes e as datas de nascimento de todos os clientes que nasceram em 05 de Fevereiro de 1969 ('Fevereiro 05, 1969').

```
SELECT nome, dt_nascimento
FROM tclientes
WHERE dt_nascimento = TO_DATE('05/02/1969', 'DD/MM/YYYY');
```

NOME	DT_NASCIMENTO
José Batista	05/02/69

Utilizando a Função CAST

Sintaxe:

```
CAST (x AS type)
```

Você pode usar a função CAST para converter *x* para um tipo de dado compatível especificado por *type*.

Exemplo:

```
SELECT CAST(preco AS VARCHAR2(10))
       , CAST(preco * 1.10 AS NUMBER(7,2))
       , CAST(preco AS BINARY_DOUBLE)
FROM tcursos;
```

CAST(PRECOASVARCHAR2(10))	CAST(PRECO*1.10ASNUMBER(7,2))	CAST(PRECOASBINARY_DOUBLE)
1800	1980	1800.0
1800	1980	1800.0
1275	1402,5	1275.0
1275	1402,5	1275.0
956	1051,6	956.0
1000	1100	1000.0
1000	1100	1000.0
956	1051,6	956.0
800	880	800.0
956	1051,6	956.0

A tabela a seguir mostra as conversões válidas (conversões válidas estão marcadas com um X).

Para	De						
	BINARY_FLOAT BINARY_DOUBLE	CHAR VARCHAR2	NUMBER	DATE TIMESTAMP INTERVAL	RAW	ROWID UROWID	NCHAR NVARCHAR2
BINARY_FLOAT BINARY_DOUBLE	X	X	X				X
CHAR VARCHAR2	X	X	X	X	X	X	
NUMBER	X	X	X				X
DATE TIMESTAMP INTERVAL		X		X			
RAW		X			X		
BINARY_ROWID UROWID		X				X	
NCHAR NVARCHAR2	X		X	X	X	X	X

Função NVL

```
NVL(desconto,0)
NVL(dt_nascimento,'01/01/70')
NVL(nome,'Nome não cadastrado!')
```

Para converter um valor nulo para outro valor do mesmo tipo, utilize a função NVL

```
NVL(expr1, expr2)
```

Sintaxe:

Onde:

expr1: é o valor ou expressão de origem que pode conter nulo.

expr2: é o valor de destino utilizado quando o valor de origem for nulo.

Você pode utilizar a função NVL para converter qualquer tipo de dado, porém, o valor de retorno deve sempre ser do mesmo tipo de dado do parâmetro *expr1*.

Função NVL2

Podemos utilizar como alternativa a função NVL2, onde definimos também um valor para quando o valor ou expressão de origem conter nulo. Para converter um valor nulo para outro valor do mesmo tipo, utilize a função NVL2.

Sintaxe:

NVL2(*expr1*, *expr2*, *expr3*)

Onde:

expr1: é o valor ou expressão de origem que pode conter nulo.

expr2: é o valor caso *expr1* não seja nula.

expr3: é o valor de destino utilizado quando o valor de origem for nulo.

Utilizando a Função NVL e NVL2

```
SELECT id, desconto, total, NVL(desconto,0) nv1
      , NVL2(desconto,total*.1,0) nv12
FROM tcontratos;
```

ID	DESCONTO	TOTAL	NVL	NVL2
1000	360	3600	360	360
1001	225	4500	225	450
1002	(null)	1800	0	0
1003	985	1995	985	199,5
1004	(null)	9000	0	0
1005	191	3191	191	319,1
1006	(null)	2000	0	0
1007	50	1000	50	100
1008	700	6218	700	621,8
1009	(null)	1200	0	0

Para calcular o desconto de todos os contratos, você precisa do valor no campo desconto. Mas e se esse valor for nulo?

Para mostrar os valores de descontos iguais a 0 nos contratos com descontos nulos utilizamos a função NVL, alternativamente podemos utilizar a função NVL2 para assumir que 10% do valor do contrato é o desconto aplicado.

Para calcular o desconto de todos os contratos, você precisa do valor no campo desconto. Mas e se esse valor for nulo?

Para mostrar os valores de descontos iguais a 0 nos contratos com descontos nulos utilizamos a função NVL, alternativamente podemos utilizar a função NVL2 para assumir que 10% do valor do contrato é o desconto aplicado.

Utilizando a Função NULLIF

Sintaxe:

```
NULLIF (expr1, expr2)
```

Onde:

NULLIF compara as expressões (expr1 e expr2). Se elas são iguais, então a função retorna nulo. Se elas são diferentes, então a função retorna expr1.

Obs.: Você não pode especificar NULL para expr1.

A função NULLIF é logicamente equivalente a seguinte expressão CASE:

```
CASE
  WHEN expr1 = expr 2 THEN NULL
  ELSE expr1
END
```

Exemplo:

```
SELECT id, desconto, total * 0.2, NULLIF (desconto, total * 0.2) RESULTADO
FROM tcontratos;
```

R Z	ID	R Z	DESCONTO	R Z	TOTAL*0.2	R Z	RESULTADO
	1000		360		720		360
	1001		225		900		225
	1002		(null)		360		(null)
	1003		985		399		985
	1004		(null)		1800		(null)
	1005		191		638,2		191
	1006		(null)		400		(null)
	1007		50		200		50
	1008		700		1243,6		700
	1009		(null)		240		(null)

Utilizando a Função COALESCE

Sintaxe:

```
COALESCE (expr1, expr2 [, expr3..])
```

Onde:

COALESCE retorna a primeira expressão diferente de nulo na lista de expressões. Você deve especificar pelo menos duas expressões. Se todas as expressões retornarem nulo, então a função retorna nulo.

Exemplo:

```
SELECT nome, cidade, estado,
       COALESCE(cidade, estado,
                'Sem cidade e estado')
FROM tclientes;
```

NOME	CIDADE	ESTADO	COALESCE(CIDADE,ESTADO,'SEMCIDADEEEESTADO')
Amarildo Fagundes	São Paulo	SP	São Paulo
Ana Maria Prado	Rio de Janeiro	RJ	Rio de Janeiro
Antônio da Silva	São Paulo	SP	São Paulo
Ramiro Antunes	Rio de Janeiro	RJ	Rio de Janeiro
Nadia Velasques	Rio de Janeiro	RJ	Rio de Janeiro
Alfredo Fonseca	Porto Alegre	RS	Porto Alegre
Jânio Marques	São Paulo	SP	São Paulo
José Batista	Porto Alegre	RS	Porto Alegre
Carlos Magno	São Paulo	SP	São Paulo
João Medeiros	Rio de Janeiro	RJ	Rio de Janeiro

Uso de CASE no SELECT

Sintaxe:

```
CASE expr
  WHEN expr_comparação1 THEN retorno1
  [WHEN expr_comparação2 THEN retorno2
  WHEN expr_comparação3 THEN retorno3
  ELSE retorno]
END;
```

Ou

```
CASE
  WHEN expr_comparação1 THEN retorno1
  [WHEN expr_comparação2 THEN retorno2
  WHEN expr_comparação3 THEN retorno3
  ELSE retorno]
END;
```

O uso de uma expressão envolvendo a cláusula CASE permite uma estrutura do tipo IF..THEN...ELSE dentro de um comando de SQL sem que tenhamos de usar uma rotina.

```
SELECT estado, CASE estado WHEN 'RS' THEN 'Rio Grande do Sul'
                        WHEN 'RJ' THEN 'Rio de Janeiro'
                        WHEN 'SP' THEN 'São Paulo'
                        ELSE ' '
                    END NOME
FROM tclientes;
```

ESTADO	NOME
SP	São Paulo
RJ	Rio de Janeiro
SP	São Paulo
RJ	Rio de Janeiro
RJ	Rio de Janeiro
RS	Rio Grande do Sul
SP	São Paulo
RS	Rio Grande do Sul
SP	São Paulo
RJ	Rio de Janeiro

Neste exemplo analisamos o valor retornado pela coluna ESTADO e de acordo com o conteúdo retornamos informações diferentes linha a linha. Esta sintaxe é similar ao uso do

DECODE; no entanto, podemos usar uma segunda forma sintática que elimina a necessidade da comparação e na qual podemos definir diversas condições diferentes.

Função DECODE

```
DECODE(coluna/expressão, arg1, result1  
      [, arg2, result2,...,]  
      [, default])
```

A função DECODE decodifica uma expressão de modo semelhante a lógica IF-THEN-ELSE utilizada em diversas linguagens. A função DECODE decodifica a expressão após compará-la com cada valor de pesquisa. Se a expressão for igual ao valor de pesquisa, o resultado correspondente é retornado.

Se o valor default for omitido, um valor nulo será retornado quando o parâmetro *col/expression* não for igual a nenhum dos valores de pesquisa.

Utilizando a Função DECODE

```
SELECT id, nome, estado,  
       DECODE(estado, 'RS', 'Gaúcho',  
              'SP', 'Paulista',  
              'Brasileiro') CLIENTES  
FROM tclientes;
```

ID	NOME	ESTADO	CLIENTES
100	Amarildo Fagundes	SP	Paulista
110	Ana Maria Prado	RJ	Brasileiro
120	Antônio da Silva	SP	Paulista
130	Ramiro Antunes	RJ	Brasileiro
140	Nadia Velasques	RJ	Brasileiro
150	Alfredo Fonseca	RS	Gaúcho
160	Jânio Marques	SP	Paulista
170	José Batista	RS	Gaúcho
180	Carlos Magno	SP	Paulista
190	João Medeiros	RJ	Brasileiro

No comando SQL acima, o valor de ESTADO é decodificado. Se ESTADO for RS, o retorno é 'Gaúcho'; se o ESTADO for SP, o retorno é 'Paulista'. Para todos os outros estados, o retorno é 'Brasileiro'.

Aninhando Funções

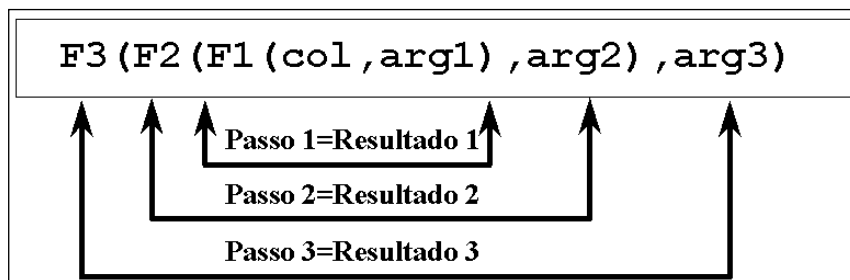


Figura 4-10: Aninhando funções

Funções básicas (single-row) podem ser aninhadas em qualquer nível. Funções aninhadas são avaliadas do nível mais interno para o nível mais externo. Abaixo seguem alguns exemplos para mostrar a flexibilidade destas funções.

```
SELECT cod_trg CURSO,
       NVL(TO_CHAR(pre_requisito), 'Sem Pré-Requisito') STATUS
FROM tcursos
WHERE pre_requisito IS NULL;
```

CURSO	STATUS
TO10G1	Sem Pré-Requisito
TSQLST	Sem Pré-Requisito
TMDPBD	Sem Pré-Requisito
TD31A	Sem Pré-Requisito
TD31U	Sem Pré-Requisito
TOOUMI	Sem Pré-Requisito
TIL	Sem Pré-Requisito
TLAPAC	Sem Pré-Requisito
TSRF	Sem Pré-Requisito
TSRFS	Sem Pré-Requisito

O exemplo acima exibe os cursos sem pré-requisitos. A avaliação do comando SQL envolve dois passos:

1. Avalia a função interna para converter um valor numérico para uma string de caracteres.

Resultado1 = TO_CHAR(pre_requisito)

2. Avalia a função externa para substituir o valor nulo por uma string de texto.
- NVL(Resultado1, 'Sem Pré-Requisito')

Exercícios – 4

1. Conecte-se ao banco de dados com o SQL Developer.
2. Escreva uma consulta para exibir a data atual no formato 'DD/MM/YYYY HH24:MI:SS'. Coloque o alias de coluna como "Data". Execute a consulta.
3. Mostre o id do curso, o código Target (cod_trg), o preço e o preço com um aumento de 15%. Selecione somente os cursos que possuam o número 1 em qualquer parte do id. Coloque o alias da coluna como "Novo Preço". Execute a consulta.
4. Altere a consulta anterior para adicionar uma nova coluna que subtraia o preço antigo do novo preço. Coloque o alias da coluna como "Aumento". Execute a consulta.
5. Mostre o nome de cada cliente e calcule o número de meses entre a data atual e a data de nascimento. Coloque o alias da coluna como "Meses de Vida". Ordene o resultado pelo número de meses. Arredonde o número de meses para o número inteiro mais próximo. Execute a consulta.
6. Crie uma consulta para exibir o código Target e o preço de todos os cursos. Formate o preço para exibir como R\$1.000,00. Coloque o alias da coluna como "Valor Curso". Execute a consulta.

Espaço para anotações

5. Exibindo Dados a Partir de Múltiplas Tabelas

Objetivos

- Escrever comandos SELECT para acessar dados de mais de uma tabela utilizando diversos tipos de JOINS;
- Visualizar dados que geralmente não correspondem a condição de JOIN utilizando OUTER JOINS;
- Executar um JOIN de uma tabela com ela mesma (SELF JOIN).

Obtendo Dados a Partir de Múltiplas Tabelas

Às vezes você precisa utilizar dados de mais de uma tabela. No exemplo a seguir, o relatório exibe dados de duas tabelas diferentes.

- ID (código cliente) existe nas tabelas TCLIENTES.ID e TCONTRATOS.TCLIENTES_ID.
- ID (código contrato) existe na tabela TCONTRATOS.ID.
- NOME (nome do cliente) existe na tabela TCLIENTES.nome.

Para produzir o relatório, você precisa unir as tabelas TCLIENTES e TCONTRATOS e acessar os dados a partir de ambas.

Tabela TCLIENTES

Column Name	Data Type
ID	NUMBER(6,0)
NOME	VARCHAR2(35 BYTE)
DT_NASCIMENTO	DATE
ENDERECO	VARCHAR2(40 BYTE)
CIDADE	VARCHAR2(30 BYTE)
ESTADO	VARCHAR2(2 BYTE)
CEP	VARCHAR2(8 BYTE)
TELEFONE	VARCHAR2(10 BYTE)
COMENTARIOS	VARCHAR2(1000 BYTE)

Tabela TCONTRATOS

Column Name	Data Type
ID	NUMBER(4,0)
DT_COMPRA	DATE
STATUS	VARCHAR2(1 BYTE)
TCLIENTES_ID	NUMBER(6,0)
DESCONTO	NUMBER(7,2)
TOTAL	NUMBER(10,2)

Figura 5-1: Descrição das tabelas TCLIENTES e TCONTRATOS

O que é um Join?

Sintaxe:

```
SELECT tabela1.coluna, tabela2.coluna [, tabela2.coluna...]  
FROM   tabela1, tabela2  
WHERE  tabela1.coluna1 = tabela2.coluna2;
```

Quando dados de mais de uma tabela são requeridos, uma condição de JOIN é utilizada. Linhas em uma tabela podem ser unidas à linhas em outra tabela de acordo com valores comuns que existem em colunas correspondentes, que normalmente são colunas de chaves primárias e estrangeiras.

Para exibir dados de duas ou mais tabelas relacionadas, escreva uma condição de JOIN simples na cláusula WHERE.

Sintaxe:

tabela.coluna: denota a tabela e coluna a partir da qual os dados são recuperados.

Tabela1.coluna1 = tabela2.coluna2 é a condição que une (ou relaciona) as tabelas.

Diretrizes

- Quando escrever um comando SELECT que relaciona tabelas, preceda o nome das colunas com o nome da tabela para obter maior clareza e melhorar o acesso ao banco de dados.
- Se o mesmo nome de coluna existir em mais de uma tabela, o nome de coluna deve ser prefixado com o nome da tabela.
- Para unir n tabelas, você precisa de um mínimo de (n-1) condições de join. Portanto, para unir quatro tabelas, um mínimo de três joins são necessários. Esta regra pode não se aplicar se a tabela possuir uma chave primária composta. Neste caso mais de uma coluna é necessária para identificar cada linha de forma exclusiva.

Produto Cartesiano

Quando uma condição de JOIN é inválida ou completamente omitida, o resultado é um produto cartesiano no qual serão exibidas todas as combinações das linhas. Todas as linhas da primeira tabela são unidas a todas as linhas da segunda tabela.

Um produto cartesiano tende a gerar um número grande de linhas, e seu resultado é raramente útil. Você sempre deveria incluir uma condição de JOIN válida na cláusula WHERE, a menos que você tenha uma necessidade específica para combinar todos os registros de todas as tabelas.

Gerando um Produto Cartesiano

Um produto cartesiano é gerado se uma condição de JOIN for omitida. O exemplo no gráfico acima exibe o nome do cliente e os descontos que este possui nos contratos de cursos estabelecidos. Uma vez que nenhuma cláusula WHERE foi especificada, todas as linhas (21 linhas) da tabela TCONTRATOS são unidas com todas as linhas (11 linhas) da tabela TCLIENTES, gerando um total de 231 linhas na consulta.

```
SELECT  nome, desconto
FROM    tclientes, tcontratos;
```

A 2	NOME	A 2	DESCONTO
	Amarildo Fagundes		360
	Amarildo Fagundes		225
	Amarildo Fagundes		(null)
	Amarildo Fagundes		985
	Amarildo Fagundes		(null)
	Amarildo Fagundes		191
	Amarildo Fagundes		(null)
	Amarildo Fagundes		50
	Amarildo Fagundes		700
	Amarildo Fagundes		(null)

Tipos de Joins

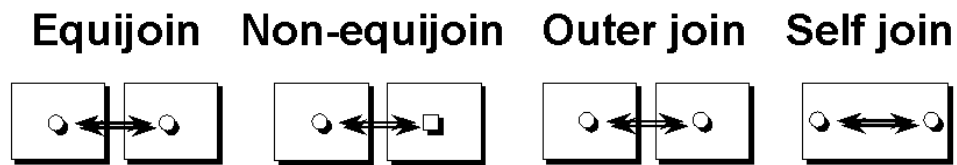


Figura 5-2: Tipos de Joins

Existem dois tipos principais de condições de join:

- Equijoins
- Non-equijoins

Métodos adicionais de JOIN incluem o seguinte:

- Outer joins
- Self joins

O que é um Equijoin?

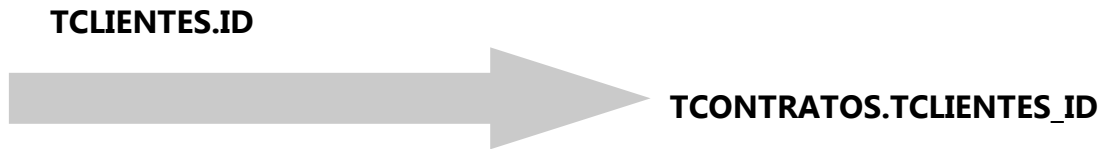


Figura 5-3: Equijoin entre as tabelas TCLIENTES e TCONTRATOS

Para determinar o nome do cliente de um contrato, você compara o valor na coluna ID da tabela TCLIENTES com os valores de TCLIENTES_ID da tabela TCONTRATOS. A relação entre as tabelas TCLIENTES e TCONTRATOS é um equijoin, ou seja, os valores na coluna ID e TCLIENTES_ID em ambas as tabelas devem ser iguais. Frequentemente, este tipo de JOIN envolve complementos de chave primária e estrangeira.

Nota: Equijoins também são chamados de joins simples ou de inner joins.

Recuperando Registros com Equijoins

```
SELECT  tclientes.id CODIGO_CLIENTE, tclientes.nome NOME,
        tcontratos.id CODIGO_CONTRATO,
        tcontratos.total TOTAL
FROM    tclientes, tcontratos
WHERE   tclientes.id = tcontratos.tclientes_id;
```

Ou

```
SELECT  tcontratos.id, tcontratos.dt_compra,
        tcontratos.total,
        tclientes.id cliente, tclientes.nome
FROM    tclientes
        JOIN tcontratos
        ON (tclientes.id = tcontratos.tclientes_id);
```

CODIGO_CLIENTE	NOME	CODIGO_CONTRATO	TOTAL
100	Amarildo Fagundes	1001	4500
100	Amarildo Fagundes	1013	2600
100	Amarildo Fagundes	1008	6218
110	Ana Maria Prado	1018	1912
110	Ana Maria Prado	1009	1200
120	Antônio da Silva	1002	1800
130	Ramiro Antunes	1010	2215
130	Ramiro Antunes	1017	3600
140	Nadia Velasques	1011	3000
150	Alfredo Fonseca	1000	3600

No exemplo acima:

- A cláusula SELECT especifica os nomes de coluna a recuperar:
 - nome do cliente e o código do cliente são colunas da tabela TCLIENTES.
 - número do contrato e o código do cliente que são colunas da tabela TCONTRATOS.
- A cláusula FROM especifica as duas tabelas que o banco de dados deve acessar:
 - Tabela TCLIENTES.
 - Tabela TCONTRATOS.
- A cláusula WHERE especifica como as tabelas serão unidas:
 - TCLIENTES.ID = TCONTRATOS.TCLIENTES_ID

Qualificando Nomes de Colunas Ambíguos

Você precisa qualificar os nomes das colunas na cláusula WHERE com o nome da tabela para evitar ambiguidade. Sem os prefixos de tabela, colunas com nomes iguais e em tabelas diferentes podem causar confusão. É necessário adicionar o prefixo de tabela para executar a consulta.





Se não existir nenhum nome de coluna comum entre as duas tabelas, não há necessidade de qualificar as colunas. Entretanto, você ganhará desempenho utilizando o prefixo de tabela porque você informa exatamente para o Servidor Oracle onde procurar pelas colunas.

A exigência para qualificar os nomes de coluna ambíguos também é aplicável para colunas que podem ser ambíguas em outras cláusulas como SELECT ou ORDER BY.

Condições Adicionais de Pesquisa com o Operador AND

Em adição ao join, você pode ter critérios adicionais na cláusula WHERE. Por exemplo, para exibir o número do cliente, o nome, o código dos contratos e o total, apenas para o cliente 'Carlos Magno', você precisa de uma condição adicional na cláusula WHERE.

```
SELECT  tclientes.id CODIGO_CLIENTE, tclientes.nome NOME,
        tcontratos.id CODIGO_CONTRATO,
        tcontratos.total TOTAL
FROM    tclientes, tcontratos
WHERE   (tclientes.id = tcontratos.tclientes_id) AND
        (UPPER(tclientes.nome) = 'CARLOS MAGNO');
```

 CODIGO_CLIENTE	 NOME	 CODIGO_CONTRATO	 TOTAL
180	Carlos Magno	1004	9000
180	Carlos Magno	1020	4000

Utilizando Alias de Tabela

```
SELECT  cl.id CODIGO_CLIENTE, cl.nome NOME,  
        co.id CODIGO_CONTRATO,  
        co.total TOTAL  
FROM    tclientes cl, tcontratos co  
WHERE   cl.id = co.tclientes_id;
```

Qualificar os nomes de coluna com os nomes de tabela pode consumir muito tempo, particularmente se os nomes de tabelas forem longos. Você pode utilizar alias de tabela em vez de nomes de tabela. Da mesma maneira que um alias de coluna fornece outro nome para uma coluna, um alias de tabela fornece outro nome para uma tabela. Alias de tabela ajudam a manter o código SQL menor, utilizando menos memória.

Observe que no exemplo os alias de tabela são identificados na cláusula FROM. O nome da tabela é especificado por completo, seguido por um espaço e então pelo alias de tabela. A tabela TCLIENTES recebeu o alias CL, enquanto que a tabela TCONTRATOS recebeu o alias CO.

Diretrizes:

- Alias de tabelas podem ter até 30 caracteres de tamanho, porém, quanto menor melhor.
- Se um alias de tabela for utilizado para um nome de tabela específico na cláusula FROM, então este alias de tabela deve ser utilizado para substituir o nome da tabela em todo o comando SELECT.
- Alias de tabelas devem ser significativos.
- O alias de tabela só é válido para o comando SELECT no qual foi declarado.

Relacionando várias Tabelas

Exemplo: Join das tabelas TCONTRATOS, TCLIENTES, TITENS e TCURSOS

```
SELECT  co.id, co.dt_compra, co.total,
        cl.id, cl.nome,
        it.seq, it.qtde,
        cs.id, cs.nome
FROM    tcontratos co,
        tclientes cl,
        titens it,
        tcursos cs
WHERE   (co.tclientes_id = cl.id) AND
        (co.id = it.tcontratos_id) AND
        (it.tcursos_id = cs.id);
```

Ou

```
SELECT co.id, co.dt_compra, co.total,
        cl.id, cl.nome,
        it.seq, it.qtde,
        cs.id, cs.nome
FROM tcontratos co
     JOIN tclientes cl ON (co.tclientes_id =
cl.id)
     JOIN titens it ON (co.id =
it.tcontratos_id)
     JOIN tcursos cs ON (it.tcursos_id = cs.id);
```

ID	DT_COMPRA	TOTAL	ID_1	NOME	SEQ	QTDE	ID_2	NOME_1
1017	10/01/05	3600	130	Ramiro Antunes	1	1	1	Introdução ao Oracle 10G I: Conceitos básicos do Oracle
1004	05/01/05	9000	180	Carlos Magno	1	1	1	Introdução ao Oracle 10G I: Conceitos básicos do Oracle
1000	03/01/05	3600	150	Alfredo Fonseca	1	1	1	Introdução ao Oracle 10G I: Conceitos básicos do Oracle
1017	10/01/05	3600	130	Ramiro Antunes	2	1	2	Introdução ao Oracle 10G II: Linguagem PL/SQL, Procedi
1004	05/01/05	9000	180	Carlos Magno	2	1	2	Introdução ao Oracle 10G II: Linguagem PL/SQL, Procedi
1000	03/01/05	3600	150	Alfredo Fonseca	2	1	2	Introdução ao Oracle 10G II: Linguagem PL/SQL, Procedi
1020	12/01/05	4000	180	Carlos Magno	1	1	45	SQL Standard
1006	06/01/05	2000	200	Mário Cardoso	1	1	45	SQL Standard
1020	12/01/05	4000	180	Carlos Magno	2	1	6	SQL Standard Avançado
1006	06/01/05	2000	200	Mário Cardoso	2	1	6	SQL Standard Avançado

Non-Equijoins

Contratos que possuam uma certa faixa de desconto.

Tabela TCONTRATOS

Column Name	Data Type
ID	NUMBER(4,0)
DT_COMPRA	DATE
STATUS	VARCHAR2(1 BYTE)
TCLIENTES_ID	NUMBER(6,0)
DESCONTO	NUMBER(7,2)
TOTAL	NUMBER(10,2)

Tabela TDESCONTOS

Column Name	Data Type
CLASSE	VARCHAR2(2 BYTE)
BASE_INFERIOR	NUMBER(7,2)
BASE_SUPERIOR	NUMBER(7,2)

Figura 5-5: Descrição das tabelas TCONTRATOS e TDESCONTOS

Uma relação entre a tabela TCONTRATOS e a tabela TDESCONTOS pode ser definida como um non-equijoin, significando que nenhuma coluna da tabela TCONTRATOS corresponde diretamente a uma coluna da tabela TDESCONTOS. A relação entre as duas tabelas é a coluna DESCONTO da tabela TCONTRATOS está entre os valores definidos nos campos BASE_INFERIOR e BASE_SUPERIOR da tabela TDESCONTOS. A relação é obtida utilizando um operador diferente de igual (=), como por exemplo o operador BETWEEN.

Recuperando Registros com Non-Equijoins

```
SELECT co.id CONTRATO, co.desconto,
       de.classe, de.base_inferior, de.base_superior
FROM   tcontratos co, tdescontos de
WHERE  NVL(co.desconto,0) BETWEEN de.base_inferior
      AND de.base_superior;
```

CONTRATO	DESCONTO	CLASSE	BASE_INFERIOR	BASE_SUPERIOR
1020	(null) A		0	999
1018	(null) A		0	999
1002	(null) A		0	999
1017	(null) A		0	999
1004	(null) A		0	999
1010	(null) A		0	999
1006	(null) A		0	999
1015	(null) A		0	999
1014	(null) A		0	999
1009	(null) A		0	999

O exemplo acima cria um non-equijoin para avaliar o nível de desconto de um contrato. O desconto deve estar entre qualquer faixa de valor de menor e maior valor.

É importante observar que todos os contratos aparecem precisamente uma única vez quando a consulta é executada. Nenhum contrato é repetido na lista. Existem duas razões para isto:

- Nenhuma das linhas da tabela de níveis de desconto contém graus que se sobrepõem. Ou seja, o valor do desconto de um contrato entre os valores de menor e maior de uma das linhas da tabela de níveis de desconto.
- Todos os descontos dos contratos estão dentro dos limites fornecidos pela tabela de níveis de desconto. Ou seja, nenhum contrato tem desconto menos que o menor valor contido na coluna BASE_INFERIOR ou mais que o maior valor contido na coluna BASE_SUPERIOR.

Nota: Outros operadores como ' \leq ' e ' \geq ' podem ser utilizados, porém o operador BETWEEN é o mais simples. Lembre-se de especificar o menor valor primeiro e o maior valor por último quando utilizar o operador BETWEEN. Alias de tabela foram especificados por razões de desempenho, não por causa de possíveis ambiguidades. Utilizamos a função NVL para garantir a comparação com valores diferentes de nulo.

Outer Joins

Como descobrir quais cursos não foram vendidos?

Tabela TITENS

Column Name	Data Type
TCONTRATOS_ID	NUMBER(4,0)
SEQ	NUMBER(4,0)
TCURSOS_ID	NUMBER(6,0)
QTDE	NUMBER(3,0)
TOTAL	NUMBER(8,2)

Tabela TCURSOS

Column Name	Data Type
ID	NUMBER(4,0)
NOME	VARCHAR2(100 BYTE)
COD_TRG	VARCHAR2(8 BYTE)
PRECO	NUMBER(7,2)
CARGA_HORARIA	NUMBER(3,0)
DT_CRIACAO	DATE
PRE_REQUISITO	NUMBER(4,0)

Figura 5-6: Descrição das tabelas TITENS E TCURSOS

Se uma linha não satisfaz a condição de join, esta linha não aparecerá no resultado da consulta. Por exemplo, na condição de equijoin das tabelas TCURSOS e TITENS, alguns cursos não foram contratados por nenhum cliente, ou seja, não existem contratos com itens de determinados cursos.

Recuperando Registros sem Correspondência Direta Utilizando Outer Joins

```
SELECT tabela1.coluna1, tabela2.coluna2  
FROM   tabela1, tabela2  
WHERE  tabela1.coluna1 (+) = tabela2.coluna2;
```

```
SELECT tabela1.coluna1, tabela2.coluna2  
FROM   tabela1, tabela2  
WHERE  tabela1.coluna1 = tabela2.coluna2 (+);
```

A(s) linha(s) sem correspondência podem ser recuperadas se um operador de OUTER JOIN for utilizado na condição de join. O operador é o sinal de adição colocado entre parênteses (+), e é colocado no "lado" do JOIN que é deficiente de informação. Este operador possui o efeito de criar uma ou mais linhas nulas, para as quais uma ou mais linhas da tabela não deficiente podem ser unidas.

Sintaxe:

tabela1.coluna1 = é a condição que relaciona (joins) as tabelas.

tabela2.coluna2 (+): é o símbolo de outer join; ele pode ser colocado em qualquer lado da condição da cláusula, porém, não pode ser colocado em ambos os lados. Coloque o símbolo de OUTER JOIN logo após o nome da coluna da tabela que pode não possuir dados para corresponder às linhas da outra tabela.

Utilizando Outer Joins

```
SELECT it.tcontratos_id, it.seq, it.qtde, cs.id, cs.nome
FROM titens it, tcursos cs
WHERE it.tcursos_id(+) = cs.id
ORDER BY it.tcontratos_id NULLS FIRST, it.seq NULLS FIRST;
```

Ou

```
SELECT it.tcontratos_id, it.seq, it.qtde, cs.id, cs.nome
FROM titens it
right outer join tcursos cs on (it.tcursos_id = cs.id)
ORDER BY it.tcontratos_id NULLS FIRST, it.seq NULLS FIRST;
```

TCONTRATOS_ID	SEQ	QTDE	ID	NOME
(null)	(null)	(null)	75	ColdFusion
(null)	(null)	(null)	39	Flash MX Avançado - Recursos avançados em Flash para criação de Web sites
(null)	(null)	(null)	41	ASP - Criando uma Loja Virtual em Banco de Dados
(null)	(null)	(null)	42	ASP Avançado
(null)	(null)	(null)	20	Oracle 10G Discover para Usuários
(null)	(null)	(null)	19	Oracle 10G Discover para Administradores
(null)	(null)	(null)	17	Desenvolvimento de Aplicações Web em PL/SQL
(null)	(null)	(null)	49	PHP
(null)	(null)	(null)	50	StarOffice I
(null)	(null)	(null)	51	StarOffice II

No exemplo acima os cursos que não possuem contrato são exibidos.

Restrições do Outer Join

O operador OUTER JOIN só pode aparecer em um dos lados da expressão, o lado que não possui informação correspondente. Ele retorna essas linhas a partir de uma tabela que não possui correspondência direta para a outra tabela.

Uma condição envolvendo um OUTER JOIN não pode utilizar o operador IN ou ser unida para outra condição pelo operador OR.

Self Joins

Como descobrir os pré-requisitos dos cursos oferecidos?

Column Name	Data Type
ID	NUMBER(4,0)
NOME	VARCHAR2(100 BYTE)
COD_TRG	VARCHAR2(8 BYTE)
PRECO	NUMBER(7,2)
CARGA_HORARIA	NUMBER(3,0)
DT_CRIACAO	DATE
PRE_REQUISITO	NUMBER(4,0)

Figura 5-7: Descrição da tabelas TCURSOS

- Às vezes você precisa relacionar uma tabela com ela mesma. Para encontrar os pré-requisitos de um certo curso você precisa unir a tabela TCURSOS com ela mesma. A coluna PRE_REQUISITO da table TCURSOS referencia o ID do curso que é pré-requisito.

Exemplo:

```
SELECT prereq.nome || ' é pré-requisito de ' || curso.nome CURSOS
FROM tcursos curso, tcursos prereq
WHERE curso.pre_requisito = prereq.id;
```

CURSOS
Introdução ao Oracle 10G I: Conceitos básicos do Oracle e SQL*PLUS, SQL e SQL Avançado é pré-requisito de Introdu
Introdução ao Oracle 10G II: Linguagem PL/SQL, Procedures, Funções, Packages e Database Trigers é pré-requisito c
Introdução ao Oracle 10G II: Linguagem PL/SQL, Procedures, Funções, Packages e Database Trigers é pré-requisito c
Introdução ao Oracle 10G II: Linguagem PL/SQL, Procedures, Funções, Packages e Database Trigers é pré-requisito c
Introdução ao Oracle 10G II: Linguagem PL/SQL, Procedures, Funções, Packages e Database Trigers é pré-requisito c
Oracle 10G Developer: Forms - Parte I é pré-requisito de Otimização de Aplicativos Oracle
Oracle 10G Developer: Forms - Parte I é pré-requisito de Oracle 10G Developer: Forms - Parte II Avançado
SQL Standard Avançado é pré-requisito de PostgreSQL: Linguagem Procedural e Unidades de Programa
Oracle 10G Designer: Modelagem de Sistemas é pré-requisito de Oracle 10G Designer: Design e Geração do Banco de D
Oracle 10G Designer: Design e Geração do Banco de Dados é pré-requisito de Oracle 10G Designer: Design e Geração

O exemplo acima relaciona a tabela TCURSOS com ela mesma. Para simular duas tabelas na cláusula FROM, existem dois alias, chamados CURSO e PRE_REQ, para a mesma tabela, TCURSOS.

Exercícios – 5

1. No SQL Developer utilize a sua conexão e conecte-se ao banco de dados do curso.
2. Escreva uma consulta para exibir o nome do cliente e os contratos (id e data de compra) que este cliente possui, ordene por nome em ordem alfabética. Execute a consulta.
3. Crie uma lista única de todos os contratos que possuem clientes com a letra 'A' (maiúscula ou minúscula) no nome, ordene por ordem alfabética. Execute a consulta.
4. Escreva uma consulta para exibir o nome do cliente, id do contrato e total, para todos os contratos que não possuem desconto (NULO ou ZERO). Execute a consulta.
5. Crie uma consulta que mostre o nome do cliente, o id e total do contrato, e a classe de desconto para todos os contratos. Ordene pela classe de desconto. Execute a consulta.
6. Mostre o id, o código e a data de criação do curso e o id, o código e a data de criação do seu curso pre_requisito do mesmo para todos os cursos. Execute a consulta.

Espaço para anotações

6. Utilizando Funções de Grupo e Formando Grupos

Objetivos

- Identificar as funções de grupo disponíveis;
- Descrever o uso de funções de grupo;
- Agrupar dados utilizando a cláusula GROUP BY;
- Incluir ou excluir grupos utilizando a cláusula HAVING.

O que são Funções de Grupo?

Diferentemente das funções básicas (single-row), as funções de grupo atuam em conjuntos de linhas para obter um resultado por grupo. Estes conjuntos podem ser a tabela inteira ou a tabela dividida em grupos.

Qual foi o maior total de contrato ?

```
SELECT max(total)
FROM tcontratos;
```

MAX(TOTAL)
15960

Tipos de Funções de Grupo

Cada uma das funções aceita um argumento. A tabela abaixo identifica as opções que podem ser utilizadas na sintaxe:

Função	Descrição
AVG([DISTINCT <u>ALL</u>] <i>n</i>)	Valor médio de <i>n</i> , ignorando valores nulos
COUNT({ * [DISTINCT <u>ALL</u>] <i>expr</i> })	Número de linhas, onde <i>expr</i> computa somente os valores diferentes de nulo. Para contar todas as linhas selecionadas utilize *, incluindo linhas duplicadas e com valores nulos
MAX([DISTINCT <u>ALL</u>] <i>expr</i>)	Valor máximo de <i>expr</i> , ignorando valores nulos
MIN([DISTINCT <u>ALL</u>] <i>expr</i>)	Valor mínimo de <i>expr</i> , ignorando valores nulos
SUM([DISTINCT <u>ALL</u>] <i>n</i>)	Soma dos valores de <i>n</i> , ignorando valores nulos
STDDEV([DISTINCT <u>ALL</u>] <i>n</i>)	Desvio padrão de <i>n</i> , ignorando valores nulos
VARIANCE([DISTINCT <u>ALL</u>] <i>n</i>)	Variância de <i>n</i> , ignorando valores nulos

Tabela 6-1: Tipos de funções de grupo

Utilizando Funções de Grupo

```
SELECT função_de_grupo(coluna | expressão)
FROM   tabela
[WHERE condição
[ORDER BY expressão];
```

Diretrizes para Utilização de Funções de Grupo

- DISTINCT faz a função avaliar somente valores não duplicados; ALL faz a função considerar todos os valores, inclusive os duplicados. O padrão é ALL e portanto não precisa ser especificado.
- Os tipos de dados para os argumentos podem ser CHAR, VARCHAR2, NUMBER ou DATE onde *expr* for especificado.
- Todas as funções de grupo exceto COUNT(*) ignoram valores nulos. Para substituir um valor nulo, utilize a função NVL.

Utilizando as Funções AVG e SUM

```
SELECT  AVG (preco) , MAX (preco) , MIN (preco) , SUM (preco)
FROM    tcursos ;
```

AVG(PRECO)	MAX(PRECO)	MIN(PRECO)	SUM(PRECO)
1026,33870967741935483870967741935483871	2240	277	63633

Você pode utilizar as funções AVG, SUM, MIN e MAX em colunas que podem armazenar dados numéricos. O exemplo acima exibe a média, o maior, o menor e a soma dos preços dos cursos.

Utilizando as Funções MIN e MAX

```
SELECT  MIN(dt_compra) , MAX(dt_compra)
FROM    tcontratos;
```

MIN(DT_COMPRA)	MAX(DT_COMPRA)
03/01/05	12/01/05

Você pode utilizar as funções MAX e MIN para qualquer tipo de dado. O exemplo acima exibe o mais recente e o mais antigo contrato.

Nota: As funções AVG, SUM, VARIANCE e STDDEV só podem ser utilizadas com tipos de dados numéricos.

Utilizando a Função COUNT

```
SELECT  COUNT(*), COUNT(ID)
FROM    tcontratos;
```

COUNT(*)	COUNT(ID)
21	21

A função COUNT possui dois formatos:

- COUNT(*)
- COUNT(*expr*)
- COUNT(*) retorna o número de linhas em uma tabela, incluindo linhas duplicadas
- COUNT(*expr*) retorna o número de linhas com valor diferente de nulo na coluna identificada por *expr*.

O exemplo acima exibe o número de contratos existentes.

O exemplo abaixo exibe o número de contratos que possuem desconto. Observe que o resultado fornece um número total de 11 linhas porque 10 contratos não possuem desconto, logo tendo o valor nulo na coluna DESCONTO.

```
SELECT COUNT(desconto)
FROM    tcontratos;
```

COUNT(DESCONTO)
11

Exemplos:


Mostre o número de estados na tabela TCLIENTES.

```
SELECT  COUNT(estado)
FROM    tclientes;
```

COUNT(ESTADO)
11

Mostre o número de estados distintos na tabela TCLIENTES.

```
SELECT COUNT(DISTINCT(estado))
FROM    tclientes;
```

 COUNT(DISTINCT(ESTADO))
3

Funções de Grupo e Valores Nulos

```
SELECT  AVG(desconto)
FROM    tcontratos;
```

```
AVG(DESCONTO)
486,4545454545454545454545454545454545
```

Todas as funções de grupo exceto COUNT(*) ignoram os valores nulos da coluna. No exemplo acima, a média é calculada baseada somente nas linhas da tabela onde um valor válido está armazenado na coluna DESCONTO. A média é calculada dividindo o desconto total por todos os contratos pelo número de contratos que recebem desconto (11).

Utilizando a Função NVL com Funções de Grupo

```
SELECT  AVG(NVL(desconto, 0))
FROM    tcontratos;
```

```
AVG(NVL(DESCONTO,0))
254,809523809523809523809523809523809524
```

A função NVL força as funções de grupo a considerarem os valores nulos no cálculo. No exemplo acima, a média é calculada baseada em todas as linhas da tabela embora existam valores nulos armazenados na coluna DESCONTO. A média é calculada dividindo o desconto total para todos os contratos pelo número total de contratos(21).

Criando Grupos de Dados

Como saber o valor total gasto pelo cliente em todos os seus contratos?

Até agora, todas as funções de grupo trataram a tabela como um grande grupo de informação. Às vezes, você precisa dividir a tabela em grupos menores. Isto pode ser feito utilizando a cláusula GROUP BY.

Criando Grupos de Dados: Cláusula GROUP BY

Você pode utilizar a cláusula GROUP BY para dividir as linhas de uma tabela em grupos. Você pode então utilizar as funções de grupo para devolver informação sumarizada para cada grupo.

```
SELECT coluna, função_de_grupo(coluna)
FROM tabela
[WHERE condição]
[GROUP BY expressão_de_grupo]
[ORDER BY expressão];
```

Sintaxe:

Expressão_de_grupo: especifica as colunas cujos valores determinam a base para o agrupamento das linhas.

Diretrizes

- Se você incluir uma função de grupo em uma cláusula SELECT, você não pode selecionar resultados individuais a menos que a coluna individual apareça na cláusula GROUP BY. Você receberá uma mensagem de erro caso não inclua a coluna na lista.
- Utilizando a cláusula WHERE, você pode excluir linhas antes de fazer a divisão dos grupos.
- Você deve incluir as colunas na cláusula GROUP BY.
- Você não pode utilizar o alias de uma coluna na cláusula GROUP BY.
- Por default, as linhas são classificadas em ordem ascendente das colunas incluídas na lista da cláusula GROUP BY. Você pode sobrepor esta ordenação utilizando a cláusula ORDER BY.

[illegible]

177

```
SELECT    tclientes_id, AVG(total)
FROM      tcontratos
GROUP BY  tclientes_id
ORDER BY  AVG(total);
```

[illegible]

Agrupando por mais de uma coluna ou expressões

Como descobrir o valor total dos contratos divididos pela data de compra e estados?

Grupos Dentro de Grupos

Às vezes existe a necessidade de visualizar resultados de grupos dentro de outros grupos. Precisamos saber como agrupar itens dentro de um contrato para descobrirmos o total dos contratos de um determinado estado..

A tabela TCONTRATOS é agrupada primeiro pela data de compra e então dentro deste agrupamento ela é agrupada pelo estado do cliente. Por exemplo, os contratos com data de compra igual a 05/01/04 dos clientes do estado do RJ se agrupam e um único resultado (valor total dos contratos) é produzido para o grupo.

Utilizando a Cláusula GROUP BY em Múltiplas Colunas

```
SELECT tcn.dt_compra DATA, tcl.estado UF, SUM(tcn.total) TOTAL
FROM   tclientes tcl, tcontratos tcn
WHERE  tcl.id = tcn.tclientes_id
GROUP  BY tcn.dt_compra, tcl.estado;
```

DATA	UF	TOTAL
06/01/05	SP	8218
11/01/05	RS	600
06/01/05	RS	1000
07/01/05	RJ	5215
12/01/05	SP	4000
04/01/05	SP	8295
08/01/05	RS	1556
09/01/05	SP	4200
03/01/05	RS	3600
05/01/05	RJ	3191
05/01/05	SP	9000
06/01/05	RJ	1200

Grupos Dentro de Grupos

Você pode retornar resultados sumarizados para grupos e subgrupos listando mais de uma coluna na cláusula GROUP BY. A ordem de classificação padrão dos resultados baseia-se na ordem das colunas da cláusula GROUP BY. A seguir é apresentado como o comando SELECT acima, contendo a cláusula GROUP BY, é avaliado:

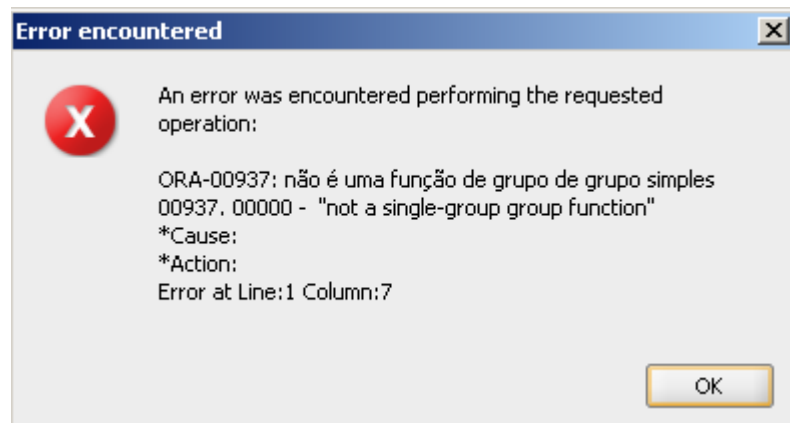
- A cláusula SELECT especifica as colunas a serem recuperadas:
- Data de Compra da tabela TCONTRATOS
- Estado da tabela TCLIENTES
- A soma do total de todos os contratos da tabela TCONTRATOS
- A soma de todos os salários para o grupo especificado na cláusula GROUP BY
- A cláusula FROM especifica as tabelas que o banco de dados deve acessar: tabelas TCONTRATOS e TCLIENTES.
- A cláusula GROUP BY especifica como as linhas devem ser agrupadas:
- Primeiro, as linhas são agrupadas através da data de compra do contrato.
- Segundo, dentro dos grupos de datas de compra do contrato, as linhas são agrupadas pelo estado do cliente.

Desta forma a função SUM está sendo aplicada à coluna de total para todos os contratos dentro de cada grupo de data de compra por estado.

Consultas Ilegais Utilizando Funções de Grupo

Em um SELECT que utiliza funções de grupo você só pode referenciar uma coluna ou expressão fora de uma função de grupo se ela estiver na cláusula "WHERE".

```
SELECT  dt_compra, COUNT(id)
FROM    tcontratos;
```



Sempre que você utilizar colunas (dt_compra) ou expressões e funções de grupo (COUNT) no mesmo comando SELECT, você deve incluir uma cláusula GROUP BY que especifique as colunas (neste caso, dt_compra) ou expressões que não estão em funções de grupo. Se a cláusula GROUP BY não for informada, então a mensagem de erro "not a single-group group function" aparece e um asterisco (*) aponta para a coluna que causou o erro. Você pode corrigir o erro acima adicionando uma cláusula GROUP BY.

Qualquer coluna ou expressão na lista da cláusula SELECT que não é argumento de uma função de grupo deve estar especificada na cláusula GROUP BY.

Exemplo:

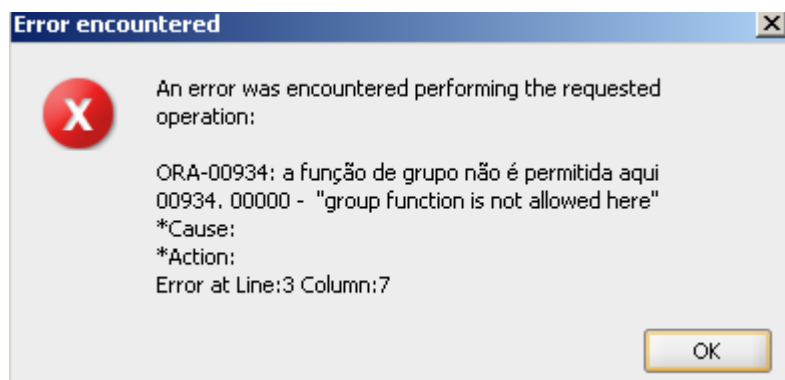
```
SELECT  dt_compra, COUNT(id)
FROM    tcontratos
GROUP BY dt_compra;
```

DT_COMPRA	COUNT(ID)
06/01/05	4
07/01/05	2
08/01/05	3
03/01/05	1
10/01/05	2
05/01/05	2
09/01/05	2
11/01/05	1
04/01/05	3
12/01/05	1

Cláusula Having

A cláusula WHERE não pode ser utilizada para restringir grupos. O comando SELECT acima resulta em um erro porque utiliza a cláusula WHERE para restringir a exibição das médias de totais de contratos por clientes que possuem média de contratos maior que R\$5000.

```
SELECT  tclientes_id CLIENTE, AVG(total) MEDIA
FROM    tcontratos
WHERE   AVG(total) > 5000
GROUP  BY tclientes_id;
```



Você pode corrigir este erro utilizando a cláusula HAVING para restringir grupos.

```
SELECT  tclientes_id CLIENTE, AVG(total) MEDIA
FROM    tcontratos
GROUP  BY tclientes_id
HAVING  AVG(total) > 5000;
```

CLIENTE	MEDIA
180	6500
150	6720

Selecionando Grupos utilizando a cláusula Having

Como descobrir qual estado possui pelo menos um cliente com mais de 40 anos?

Da mesma forma que você utiliza a cláusula WHERE para restringir as linhas selecionadas, você utiliza a cláusula HAVING para restringir grupos.

- Encontrar a maior idade para cada estado agrupando pelo campo ESTADO.
- Restringir os grupos para esses estados, listando somente os que tiverem um cliente com a idade maior que 40 anos.

```
SELECT  coluna, função_de_grupo(coluna |
expressão)
FROM    tabela
[WHERE  condição]
[GROUP BY expressão_de_grupo]
[HAVING condição_de_grupo]
[ORDER BY coluna];
```

Você utiliza a cláusula HAVING para especificar quais grupos serão exibidos. Portanto, você restringe grupos baseado em informações agregadas.

Sintaxe:

Condição_de_grupo: restringe as linhas de grupos retornadas para aqueles grupos onde a condição especificada retornar TRUE.

O Servidor Oracle executa os seguintes passos quando você utiliza grupos e a cláusula HAVING:

1. A cláusula WHERE seleciona as linhas.
2. A cláusula GROUP BY forma os grupos.
3. A cláusula HAVING seleciona os grupos.

Exemplo:

```
SELECT estado, MIN(dt_nascimento)
FROM    tclientes
GROUP BY estado
HAVING MIN(dt_nascimento) <
(ADD_MONTHS(sysdate, -1 * (40*12)));
```

ESTADO	MIN(DT_NASCIMENTO)
RJ	19/10/61
SP	10/07/64
RS	05/02/69

O exemplo acima exibe os estados que possuem clientes com idade superior a 40 anos.

O exemplo abaixo exibe o número do contrato e a média do total para os contratos cujo total seja maior que R\$5000 por cliente.

```
SELECT tcclientes_id,  
AVG(total)  
FROM   tcontratos  
GROUP  BY tcclientes_id  
HAVING MAX(total) > 5000;
```

[illegible]

O exemplo abaixo exhibe os contratos agrupados por clientes onde a diferença entre a data de compra e a data atual seja inferior a 8 dias (uma semana) e a soma do total destes contratos seja superior a R\$3000.

```
SELECT dt_compra DT_COMPRA, SUM(desconto)
DESCONTO
FROM      tcontratos
WHERE     sysdate - dt_compra < 8
GROUP BY  dt_compra
HAVING    SUM(desconto) > 3000
ORDER BY  DESCONTO;
```

Aninhando Funções de Grupo

```
SELECT MAX(AVG(total))  
FROM   tcontratos  
GROUP BY tclientes_id;
```

MAX(AVG(TOTAL))
6720

Funções de grupo podem ser aninhadas. O exemplo acima exibe a maior média de contrato por cliente.

Exercícios – 6

1. Funções de grupo atuam sobre muitas linhas para produzir uma única linha para cada grupo formado?
2. Funções de grupo incluem nulos nos cálculos?
3. No SQLDeveloper utilize a sua conexão e conecte-se ao banco de dados do curso. Mostre o maior, o menor, a soma e a média do total de todos os contratos. Coloque o alias das colunas como "Máximo", "Mínimo", "Soma" e "Média". Arredonde os resultados com zero dígitos decimais. Execute a consulta.
4. Modifique a consulta anterior para exibir o menor, o maior, a soma e a média do total para cada estado. Execute a consulta.
5. Escreva uma consulta que mostre a diferença entre o maior e menor contrato. Coloque o alias da coluna como "DIFERENÇA". Execute a consulta.
6. Escreva uma consulta para exibir o estado, o número de contratos e a média do total de contratos para todos os clientes daquele estado. Coloque os alias de coluna como "UF", "CONTRATOS", "MÉDIA", respectivamente. Execute a consulta.

Espaço para anotações

7. Variáveis de Substituição e Variáveis de ambiente do SQL*Plus

Objetivos

- Utilizar variáveis de substituição;
- Customizar o ambiente do SQLPlus.

Variáveis de Substituição

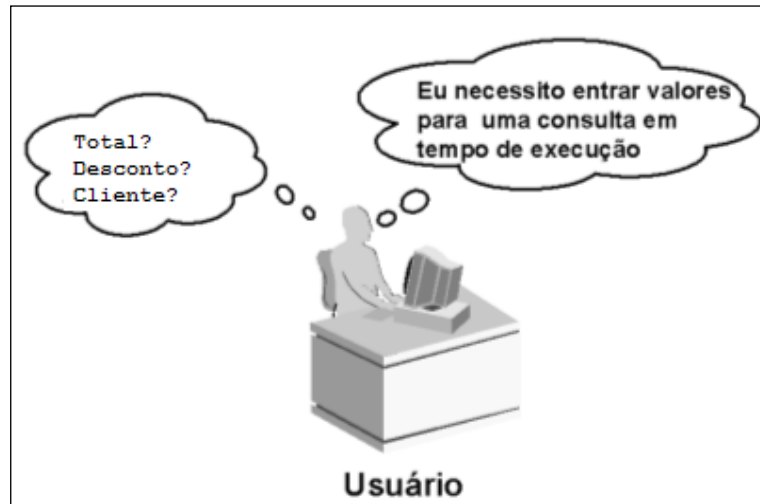
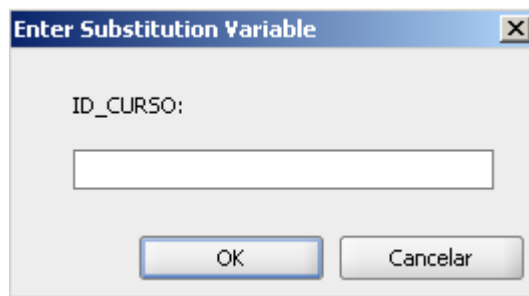


Figura 7-1: Interagindo com comandos SQL

Utilizando SQLPlus, você pode criar relatórios que solicitem ao usuário que forneça seus próprios valores para restringir o intervalo de dados retornados. Para criar relatórios interativos, você pode embutir variáveis de substituição em um arquivo de comandos ou em um comando SQL isolado. Uma variável pode ser vista como um recipiente no qual os valores são armazenados temporariamente.

Utilizando Variáveis de Substituição com (&)

```
SELECT id, cod_trg, preco
FROM   tcursos
WHERE  id = &id_curso;
```



Quando executam um relatório, os usuários freqüentemente necessitam restringir os dados retornados dinamicamente. O SQLPlus provê esta flexibilidade por meio de variáveis de usuário. Utilize o símbolo (&) para identificar cada variável em seu comando SQL. Você não precisa definir o valor de cada variável.

Sintaxe	Descrição
<i>&variável</i>	Indica uma variável em um comando SQL; se a variável não existe, o SQL*Plus solicita ao usuário um valor (o SQL*Plus descarta uma nova variável uma vez que ela tenha sido utilizada).

Tabela 7-1: Variável de substituição

O exemplo acima cria um comando SQL para solicitar ao usuário um id do cursos em tempo de execução e exibe o número do curso, o código Target e o preço para aquele curso.

Com o uso de um único símbolo (&), o usuário é solicitado cada vez que o comando é executado, se a variável não existir.

Utilizando o Comando SET VERIFY

```
SQL> SET VERIFY ON
SQL> SELECT nome, cidade, estado
2  FROM   tclientes
3  WHERE  id = &id_cliente;
```

No SQLPlus

Para confirmar as mudanças no comando SQL, utilizar o comando do SQL*Plus SET VERIFY. Ao executar SET VERIFY ON o SQLPlus passa a exibir o texto de um comando antes e depois de efetuar a troca das variáveis de substituição pelos valores.

O exemplo acima exibe o antigo como também o novo valor da variável de substituição id_cliente.

```
Informe o valor para id_cliente: 150
antigo   3:      WHERE id = &id_cliente
novo     3:      WHERE id = 150
...
```

Valores Caractere e Data com Variáveis de Substituição

```
SELECT nome, cidade, estado  
FROM tclientes  
WHERE estado = '&pestado';
```

Em uma cláusula WHERE, valores tipo data e caractere devem ser incluídos entre aspas simples. A mesma regra aplica-se às variáveis de substituição.

Para evitar a necessidade de entrar aspas em tempo de execução, inclua a variável entre aspas simples dentro do próprio comando SQL.

O exemplo acima apresenta uma consulta para recuperar o nome do cliente, a cidade e o estado de todos os clientes baseado no nome fornecido pelo usuário no prompt.

Nota: Você também pode utilizar funções como UPPER e LOWER com o (&). Utilize UPPER('&pestado') de forma que o usuário não tenha que entrar o nome do cliente em maiúsculas.

```
SELECT nome, cidade, estado  
FROM tclientes  
WHERE UPPER(estado) = UPPER('&pestado');
```


Especificando Nomes de Colunas, Expressões e Textos em Tempo de Execução

Utilize variáveis de substituição para completar:

- Uma condição WHERE
- Uma cláusula ORDER BY
- Uma expressão de coluna
- Um nome de tabela
- Um comando SELECT inteiro

Além da cláusula WHERE dos comandos SQL, você pode utilizar as variáveis de substituição para substituir nomes de coluna, expressões ou texto.

Exemplo:

Exibe o número do cliente, o nome, e o cep e qualquer outra coluna especificada pelo usuário em tempo de execução, a partir da tabela TCLIENTES. O usuário também pode especificar a condição para recuperação de linhas e o nome da coluna pela qual os dados resultantes devem ser ordenados.

```
SELECT id, nome, cep, &nome_coluna
FROM   tclientes
WHERE  &condicao
ORDER BY &ordenacao;
```

Utilizando Variáveis de Substituição com (&&)

```
SELECT nome, cidade, estado  
FROM tclientes  
WHERE UPPER(estado) = UPPER('&&pestado');
```

Você pode utilizar variáveis de substituição com o símbolo (&&) se você quiser reutilizar o valor da variável sem solicitá-lo ao usuário cada vez.

O usuário receberá uma única vez o prompt para o valor. No exemplo acima, o usuário é solicitado a fornecer uma única vez o valor para a variável nome_coluna. O valor fornecido pelo usuário (estado) é utilizado para exibir e ordenar os dados.

O SQLPlus armazena o valor fornecido utilizando o comando DEFINE; ele o reutilizará sempre que você referenciar o nome da variável. Se necessário, você pode utilizar o comando UNDEFINE para apagar uma variável de usuário.

```
SQL> UNDEFINE variável_substituição
```

Definindo Variáveis

Você pode predefinir variáveis antes de executar um comando SELECT. O SQL*Plus provê dois comandos para definir e setar variáveis: DEFINE e ACCEPT.

Se você precisar especificar espaços quando utilizar o comando DEFINE, você deve colocar os espaços dentro de aspas simples.

Comando	Descrição
DEFINE <i>variable</i> = <i>value</i>	Cria uma variável de usuário do tipo de dado CHAR e atribui um valor a ela
DEFINE <i>variable</i>	Mostra a variável, seu valor e seu tipo de dado
DEFINE	Mostra todas as variáveis de usuário com seus valores e tipos de dados
ACCEPT (<i>veja a sintaxe a seguir</i>)	Lê uma linha de entrada do usuário e a armazena em uma variável

Tabela 7-2: Definindo variáveis de usuário

O Comando ACCEPT

Sintaxe:

```
ACCEPT variável [tipo_de_dado] [FORMAT formato]
[PROMPT texto] {HIDE}
```

variável: é o nome da variável que armazena o valor. Se ela não existir, o SQL*Plus a criará.

tipo_de_dado: deve ser NUMBER, CHAR ou DATE. CHAR possui um tamanho máximo de 240 bytes. **DATE** é verificado através de um modelo de formato, e o tipo de dado é CHAR.

FOR[MAT]: especifica a máscara de formatação, por exemplo: A10 ou 9.999.

PROMPT texto: exibe o texto antes de o usuário poder entrar o valor.

HIDE: suprime o que o usuário digita, por exemplo, uma senha.

Nota: Não prefixe o parâmetro de substituição do SQLPlus com (&) quando referenciar o parâmetro no comando ACCEPT.

```
ACCEPT pestado PROMPT 'Digite a sigla do estado: '
SELECT nome, cidade, estado
FROM tclientes
WHERE UPPER(estado) = UPPER('&pestado');
```

Utilizando o Comando ACCEPT

O comando ACCEPT lê uma variável chamada "pestado". O prompt exibido quando solicitar para o usuário a variável é "Digite a sigla do estado: ". O comando SELECT então recebe o valor do que o usuário digitou para pestado e o utiliza para recuperar a linha apropriada da tabela TCLIENTES, de acordo com o comando.

Observe que o caractere & não aparece com a variável NOME no comando ACCEPT. O & só aparece no comando SELECT.

Diretrizes:

- Ambos os comandos ACCEPT e DEFINE criam uma variável se a variável não existir; estes comandos redefinem automaticamente uma variável caso já exista.
- Quando utilizar o comando DEFINE, utilize aspas simples (') para incluir uma string que contenha espaços.
- Utilize o comando ACCEPT para:
 - Fornecer um prompt customizado quando receber entrada de usuário; caso contrário, você verá uma mensagem padrão "Enter value for variable"
 - Explicitamente defina uma variável do tipo NUMBER ou DATE
 - Oculte a entrada do usuário por razões de segurança

Comandos DEFINE e UNDEFINE

As variáveis permanecem definidas até que você:

- Execute o comando UNDEFINE para a variável
- Encerre o SQL*Plus

Quando você remove variáveis, você pode verificar suas mudanças com o comando DEFINE. Quando você encerra o SQL*Plus, as variáveis definidas durante aquela sessão são perdidas.

Utilizando o Comando DEFINE e UNDEFINE

```
DEFINE pestado = 'RS'  
SELECT nome, cidade, estado  
FROM tclientes  
WHERE UPPER(estado) = UPPER('&pestado');
```

Você pode utilizar o comando DEFINE para criar uma variável e então utilizar esta variável como você utilizaria qualquer outra variável. O exemplo acima cria uma variável pestado que contém o valor 'RS'. O comando SQL então utiliza esta variável.

Para mostrar a definição da variável pestado:

```
DEFINE pestado
```

Para apagar a variável, você utiliza o comando UNDEFINE:

```
UNDEFINE pestado
```

Variáveis de Ambiente do SQL*Plus

```
SET variável_de_ambiente valor
```

Você pode controlar o ambiente no qual SQL*Plus está operando utilizando os comandos SET.

```
SQL> SET ECHO ON
```

```
SQL> SHOW ECHO  
echo ON
```

Sintaxe:

system_variable: é uma variável que controla um aspecto do ambiente da sessão.

value: é um valor para a variável de sistema.

Você pode verificar a configuração atual com o comando SHOW. O comando SHOW no exemplo acima confere se ECHO estava configurado para ON ou para OFF.

Para ver todos os valores de variáveis SET, utilize o comando SHOW ALL.

Variáveis do Comando SET

Variáveis SET e Valores	Descrição
ECHO { <u>OFF</u> ON }	Exibe o conteúdo dos arquivos executados
FEED[BACK] { <u>6</u> <i>n</i> OFF <u>ON</u> }	Exibe o número de registros retornados por uma consulta quando a consulta retorna pelo menos <i>n</i> registros
HEA[DING] {OFF <u>ON</u> }	Determina quando os cabeçalhos de coluna devem ser exibidos nos relatórios
LIN[ESIZE] { <u>80</u> <i>n</i> }	Configura o número de caracteres por linha para <i>n</i>
PAGES[IZE] { <u>14</u> <i>n</i> }	Especifica o número de linhas por página de saída
PAU[SE] { <u>OFF</u> ON <i>text</i> }	Permite que você controle o scroll do seu terminal (Você deve pressionar [Return] após visualizar cada pausa)

Tabela 7-3: Variáveis do comando SET

Nota: O valor *n* representa um valor numérico. Os valores sublinhados apresentados acima indicam os valores default. Se você não fornecer nenhum valor com a variável, o SQLPlus assume o valor default.

Exercícios – 7

1. Conecte-se no SQL*Plus.
2. Escreva um arquivo de script para mostrar o código Target, o preço e a data de criação para todos os cursos que foram criados entre um determinado período. Concatene o código Target e data de criação, separando-os por uma vírgula e espaço, e coloque o alias da coluna como "Curso". Solicite ao usuário os dois intervalos do período utilizando o comando ACCEPT. Utilize o formato DD/MM/YYYY. Salve o script para um arquivo chamado *e7q1.sql*. Execute o script *e7q1.sql*.

Espaço para anotações

8. Sub-consultas

Objetivos

- Descrever os tipos de problemas que sub-consultas podem resolver;
- Definir sub-consultas;
- Listar os tipos de sub-consultas;
- Escrever sub-consultas do tipo single-row e multiple-row;
- Escrever uma sub-consulta multiple-column;
- Escrever sub-consultas em uma cláusula FROM.

Utilizando uma Sub-consulta para Resolver um Problema

Suponha você quer escrever uma consulta para encontrar quem tem um contrato com valor total maior que os contratos do cliente 110.

Problema:

- “Quais são os contratos maiores que o maior contrato do cliente 110?”

Para resolver este problema, você precisa de duas consultas: uma consulta para encontrar o maior valor dos contratos do cliente 110 e uma segunda consulta para encontrar quem tem contrato maior que este valor.

Você pode resolver este problema combinando as duas consultas e colocando uma consulta dentro da outra.

Uma consulta interna ou sub-consulta retorna um valor que é utilizado pela consulta externa ou consulta principal. Utilizar uma sub-consulta é equivalente a executar duas consultas sequenciais e utilizar o resultado da primeira consulta como o valor de procura da segunda consulta.

Sub-consultas

Sintaxe:

```
SELECT select_list
FROM   tabela
WHERE  expressão operador
      ( SELECT select_list
        FROM   tabela);
```

Onde:

Select_list: Lista de colunas ou expressões

operado: inclui um operador de comparação como >, = ou IN

expressão : Expressão a ser comparada com o resultado do select da sub-consulta

Uma sub-consulta é um comando SELECT embutido em uma cláusula de outro comando SELECT. Você pode construir comandos poderosos utilizando sub-consultas. Eles podem ser muito úteis quando você precisa selecionar linhas de uma tabela com uma condição que depende dos dados da própria tabela.

Você pode colocar sub-consultas em várias cláusulas SQL:

- Cláusula WHERE
- Cláusula HAVING
- Cláusula FROM

Nota: Operadores de comparação entram em duas classes: operadores do tipo single-row (>, =, >=, <, <>, <=, !=) e operadores do tipo multiple-row (IN, ANY, ALL).

A sub-consulta é frequentemente chamada de SELECT aninhado, sub-consulta, ou comando SELECT interno. A sub-consulta geralmente é executada primeiro, e seu resultado é utilizado para completar a condição da consulta principal ou externa.

Utilizando uma Sub-consulta

```
SELECT id CONTRATO
FROM   tcontratos
WHERE  total >
      ( SELECT MAX(total)
        FROM   tcontratos
        WHERE  tclientes_id = 110) ;
```

No exemplo acima, a consulta interna determina o maior total dos contratos do cliente 110. A consulta externa recebe o resultado da consulta interna e utiliza este resultado para exibir todos os contratos que possuem total maior que este valor.

Diretrizes para Utilização de Sub-consultas

- Uma sub-consulta deve ser incluída entre parênteses.
- Uma sub-consulta deve estar no lado direito do operador de comparação.
- Duas classes de operadores de comparação são utilizadas em sub-consultas: os operadores do tipo single-row e os operadores do tipo multiple-row.

Tipos de Sub-consultas

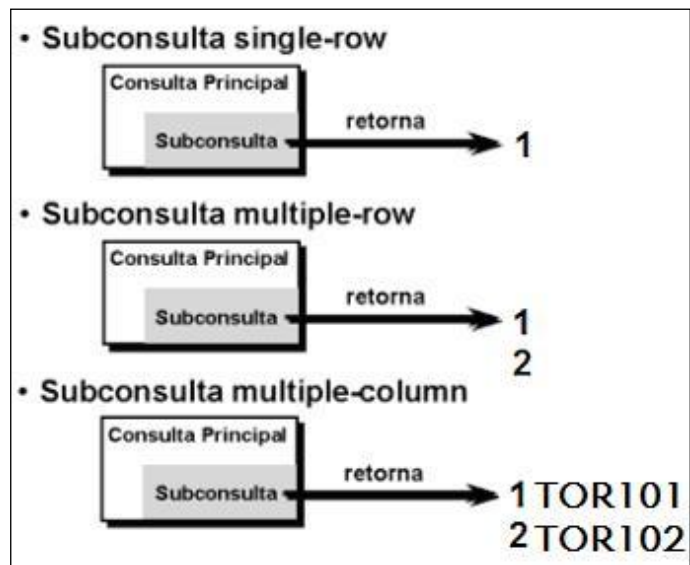


Figura 2-1: Tipos de sub-consultas

- Sub-consultas single-row: consultas que retornam apenas uma linha a partir do comando SELECT interno.
- Sub-consultas multiple-row: consultas que retornam mais de uma linha a partir do comando SELECT interno.
- Sub-consultas multiple-column: consultas que retornam mais de uma coluna a partir do comando SELECT interno.

Sub-consultas Single-Row

Operadores de comparação utilizados com Sub-consultas Single Row.

Operador	Significado
=	Igual a
>	Maior que
>=	Maior que ou igual a
<	Menor que
<=	Menor que ou igual a
<> ou !=	Diferente de

Figura 2-2: Operadores de sub-consultas single-row

Uma sub-consulta do tipo single-row retorna uma linha a partir do comando SELECT interno. O resultado deste tipo de sub-consulta deve ser comparado com uma expressão utilizando um operador do tipo single-row. A figura acima exibe uma lista dos operadores single-row.

Exemplo: Mostre os cursos cujo preço seja igual ao preço do curso com ID = 1.

```
SELECT    id, cod_trg, preco
FROM      tcursos
WHERE     preco =
          ( SELECT preco
            FROM   tcursos
            WHERE  id = 1 );
```

Multiplas Sub-consultas Single-Row

Exemplo:

```
SELECT id, nome
FROM   tclientes
WHERE  estado =
      ( SELECT estado
        FROM   tclientes
        WHERE  id = 100)
AND    TO_CHAR(dt_nascimento, 'MON') =
      ( SELECT TO_CHAR(dt_nascimento, 'MON')
        FROM   tclientes
        WHERE  id = 130);
```

Um comando SELECT pode ser considerado como um bloco de consulta. O exemplo acima exibe os clientes cujo estado é o mesmo do cliente 100 e cujo mês de nascimento é igual que o do cliente 130.

O exemplo consiste de três blocos de consulta: a consulta externa e as duas consultas internas. Os blocos de consulta internos são executados primeiro, produzindo os resultados da consulta: SP e OUT, respectivamente. O bloco de consulta externo é então processado e utiliza os valores retornados pelas consultas internas para completar as suas condições de pesquisa.

Ambas as consultas internas retornam valores únicos ('SP' e 'OUT', respectivamente), sendo chamadas de sub-consultas single-row.

Nota: As consultas externa e interna podem obter dados de tabelas diferentes.

Utilizando Funções de Grupo em uma Sub-consulta

Exemplo:

```
SELECT id, dt_compra, desconto, total
FROM   tcontratos
WHERE  total >
      ( SELECT AVG(total)
        FROM   tcontratos );
```

Você pode exibir dados de uma consulta principal utilizando uma função de grupo em uma sub-consulta para retornar uma única linha. A sub-consulta está entre parênteses e é colocada após o operador de comparação.

O exemplo acima exibe os contratos que possuem o total maior que a média do total de todos os contratos da tabela TCONTRATOS. A função de grupo AVG retorna um único valor (3530,80952) para a consulta externa.

Utilizando a cláusula Sub-consultas na cláusula HAVING

Exemplo:

```
SELECT dt_compra, AVG(total)
FROM   tcontratos
GROUP BY dt_compra
HAVING AVG(total) >
      ( SELECT AVG(total)
        FROM   tcontratos );
```

DT_COMPRA	AVG(TOTAL)
03/01/05	3600
05/01/05	6095,5
09/01/05	10080
12/01/05	4000

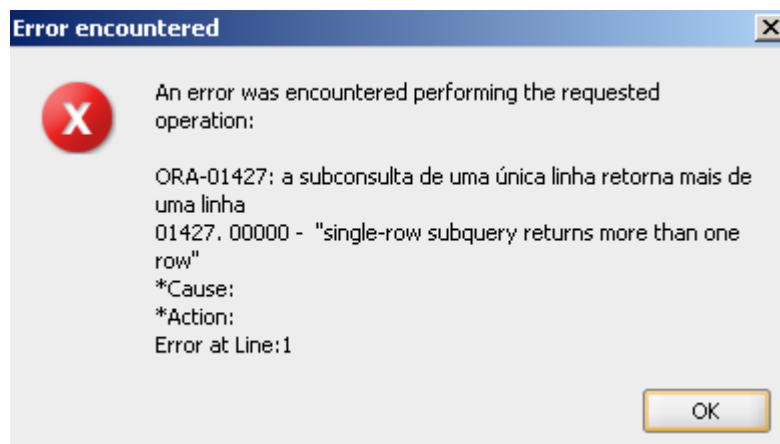
Além da cláusula WHERE, você também pode utilizar sub-consultas na cláusula HAVING. O Servidor Oracle executa a sub-consulta, e os resultados são retornados para a cláusula HAVING da consulta principal.

O comando SQL acima exibe somente os grupos formados por datas cuja media do total de contratos é superior a media do total de todos os contratos.

Erros utilizando Operador single row

Operador single row utilizado com uma Sub-consulta que retorna mais de uma linha.
Exemplo:

```
SELECT dt_compra, AVG(total)
FROM   tcontratos
GROUP BY dt_compra
HAVING AVG(total) =
        ( SELECT AVG(total)
          FROM   tcontratos
          GROUP BY dt_compra );
```



Um erro comum em sub-consultas é mais de uma linha ser retornada para uma sub-consulta do tipo single-row.

No comando SQL acima, a sub-consulta possui uma cláusula GROUP BY (dt_compra), que implica que a sub-consulta devolverá múltiplas linhas, uma para cada grupo encontrado. Neste caso, o resultado da sub-consulta será:

A consulta externa recebe os resultados da sub-consulta e utiliza estes resultados em sua cláusula WHERE. A cláusula WHERE contém um operador igual (=), operador de comparação do tipo single-row que compara apenas um valor. O operador (=) não aceita mais de um valor a partir sub-consulta e consequentemente gera o erro.

Operador single row utilizado com uma Sub-consulta que não retorna nenhuma linha.

Exemplo:

```
SELECT tclientes_id, MIN(total)
FROM   tcontratos
GROUP BY tclientes_id
HAVING AVG(total) >
        (SELECT AVG(total)
         FROM tcontratos
         WHERE tclientes_id = 500);
```

Resultado: Nenhuma linha retornada.

Um problema comum com sub-consultas é nenhuma linha ser retornada pela sub-consulta (consulta interna).

No comando SQL acima, o comando parece estar correto, mas não seleciona nenhuma linha quando é executado.

O problema é que não existem contratos para o cliente 500. Assim, a sub-consulta não retorna nenhuma linha. A consulta externa recebe os resultados da sub-consulta (null) e utiliza estes resultados na cláusula WHERE. A consulta externa não encontra nenhum cliente com a média de contratos maior do NULL (qualquer comparação com NULL retorna o booleano NULL).

Sub-consultas do Tipo Multiple-Row

Sub-consultas que retornam mais que uma linha são chamadas sub-consultas multiple-row. Você utiliza operadores multiple-row, em vez de operadores single-row, com uma sub-consulta multiple-row. O operador multiple-row aceita um ou mais valores.

Operadores de comparação utilizados com Sub-consultas Multiple Row.

Operador	Significado
IN	Igual a qualquer membro da lista
ANY	Compara um valor com cada valor retornado por uma subconsulta
ALL	Compara um valor com todos os valores retornados por uma subconsulta

Figura 2-3: Operadores de sub-consultas multiple-row

Nota:

- O Operador ANY deve ser precedido por: =, !=, >, <, <=, >=
- O Operador ALL deve ser precedido por: =, !=, >, <, <=, >=

Sub-consultas que retornam mais que uma linha são chamadas sub-consultas multiple-row. Você utiliza operadores multiple-row, em vez de operadores single-row, com uma sub-consulta multiple-row. O operador multiple-row aceita um ou mais valores.

Exemplo:

```
SELECT    id, cod_trg, preco
FROM      tcursos
WHERE     preco IN
          (SELECT MIN(preco)
           FROM    tcursos
           GROUP   BY carga_horaria);
```

Utilizando o Operador ANY em Sub-consultas Multiple-Row

```
SELECT id, dt_compra, desconto, total
FROM   tcontratos
WHERE  total > ANY
      ( SELECT total
        FROM   tcontratos
        WHERE  dt_compra = to_date('04/01/2005',
'DD/MM/YYYY') ) ;
```

O operador ANY compara um valor para cada valor retornado por uma sub-consulta. O exemplo acima exibe os contratos cujo total é menor que o total de qualquer contrato do dia '04/01/2005'. O maior total deste dia é R\$4500. O comando SQL exibe todos os contratos que tenham total menor que R\$4500.

- < ANY significa Menor do que Qualquer
- > ANY significa Maior do que Qualquer
- = ANY é equivalente a IN.

Utilizando o Operador ALL em Sub-consultas Multiple-Row

Exemplo:

```
SELECT id, dt_compra, desconto, total
FROM   tcontratos
WHERE  total > ALL
      (SELECT total
       FROM   tcontratos
       WHERE  dt_compra =
to_date('04/01/2005', 'DD/MM/YYYY'));
```

O operador ALL compara um valor com todos os valores retornados por uma sub-consulta. O exemplo acima exibe os contratos cujo total é maior que o total de todos os contratos do dia '04/01/05'. O maior total é R\$4500, assim a consulta retorna aqueles contratos cujo total é maior que R\$4500.

- < ALL significa Menor do que Todos
- > ALL significa Maior do que Todos

Nota: O operador NOT pode ser utilizado com os operadores IN, ANY e ALL.

Sub-consultas Multiple-Column

A consulta principal compara	com	Valores de uma subconsulta multiple-row e multiple-column
1 TO10G1		1 TO10G1 2 TO10G2 3 TF6I 4 TR6I

Figura 2-4: Sub-consultas multiple-column

Até agora você escreveu sub-consultas do tipo single-row e sub-consultas do tipo multiple-row onde só uma coluna foi comparada na cláusula WHERE ou na cláusula HAVING do comando SELECT. Se você quiser comparar duas ou mais colunas, você deve escrever uma combinação na cláusula WHERE utilizando os operadores lógicos. Sub-consultas do tipo multiple-column permitem combinar condições WHERE duplicadas em uma única cláusula WHERE.

Sintaxe:

```
SELECT coluna, coluna, ...
FROM   tabela
WHERE  (col1, col2, col3...) IN
      ( SELECT col1, col2, col3 ...
        FROM   tabela
        WHERE  condição );
```

Utilizando Sub-consultas Multiple-Column

Exemplo:

```
SELECT id, dt_compra, desconto, total
FROM   tcontratos
WHERE  (total, NVL(desconto,0)) IN
      ( SELECT total, NVL(desconto,0)
        FROM   tcontratos
        WHERE  total < 3000);
```

ID	DT_COMPRA	DESCONTO	TOTAL
1002	04/01/05	(null)	1800
1003	04/01/05	985	1995
1006	06/01/05	(null)	2000
1007	06/01/05	50	1000
1009	06/01/05	(null)	1200
1010	07/01/05	(null)	2215
1019	11/01/05	60	600
1012	08/01/05	60	600
1013	08/01/05	260	2600
1014	08/01/05	(null)	956
1018	10/01/05	(null)	1912

```
SELECT tcnl.id, tcnl.dt_compra, tcnl.total,      sub-consulta.media
FROM tcontratos tcnl, ( SELECT dt_compra, AVG(total) MEDIA
                        FROM tcontratos tcn2
                        GROUP BY dt_compra) sub-consulta
WHERE tcnl.dt_compra = sub-consulta.dt_compra
AND tcnl.total > sub-consulta.media;
```

[illegible]

Você pode utilizar uma sub-consulta na cláusula FROM de um comando SELECT. O exemplo acima exibe os contratos que possuem total maior que a média do total dos contratos do mesmo dia de compra (data de compra).

Cuidado com Sub-consultas que retornam NULL



Exemplo: A sub-consulta recupera valores nulos

```
SELECT cs.cod_trg, cs.preco
FROM   tcursos cs
WHERE  cs.id NOT IN
        (SELECT cs2.pre_requisito
         FROM   tcursos cs2);
```

Resultado: Nenhuma linha selecionada

Exemplo: A sub-consulta não recupera valores nulos

```
SELECT cs.cod_trg, cs.preco
FROM   tcursos cs
WHERE  cs.id NOT IN
        (SELECT cs2.pre_requisito
         FROM   tcursos cs2
         WHERE  cs2.pre_requisito
                IS NOT NULL);
```

 COD_TRG	 PRECO
TOPP	956
TPHPAV	1000
TLAPAC	665
TASPAV	1000
TF6I-A	956
TSRF	775
TSRF5	775
TD31U	638
TFLMXA	600
TOPC	956

Exercícios – 8

1. No SQL Developer utilize a sua conexão e conecte-se ao banco de dados do curso. Escreva uma consulta para exibir o nome dos clientes e a data de nascimento para todos os clientes que estão no mesmo estado do cliente 'Mário Cardoso', excluindo-o do resultado.
2. Crie uma consulta para exibir o id e o total do contrato, o nome do cliente responsável pelo contrato para todos os contratos com total maior que a média do total de contratos. Classifique o resultado em ordem descendente de total. Execute a consulta.
3. Escreva uma consulta que mostre o id e o nome do cliente para todos os clientes que são de um estado com qualquer cliente cujo nome contenha uma letra 'v' (minúscula). Execute a consulta.
4. Mostre o código Target e o preço dos cursos com pré-requisito o cod_trg 'THTML4'.
5. Mostre o nome e o telefone de todos os clientes que compraram no dia '06/01/2005'. Execute a consulta.

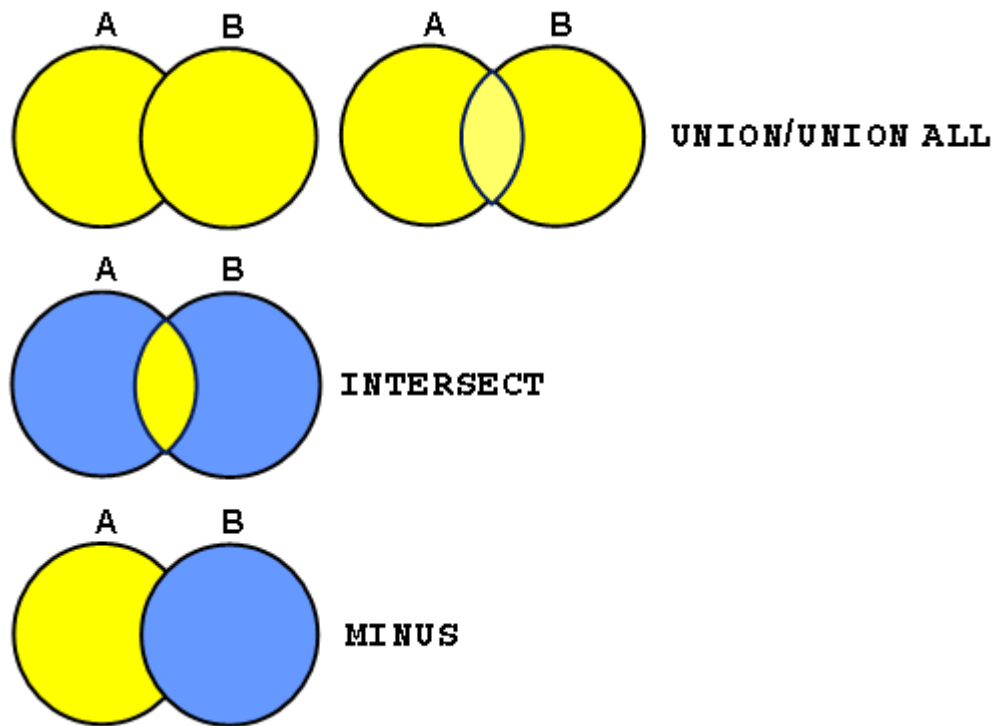
Espaço para anotações

9. Operadores SET

Objetivos

- Verificar as formas de representar os operadores SET;
- Realizar a união, intersecção e diferença entre SELECTs relacionados.

Operadores SET



União – UNION

A operação de união efetua uma soma de conjuntos eliminando as duplicidades. Considerando-se a existência das duas tabelas A e B, as linhas resultantes da união seriam representadas por LAUB. Isto significa que o resultado A é uma relação que contém apenas as colunas indicadas. As linhas duplicadas que seriam criadas são eliminadas.

```
UNION ...  
UNION ALL ...
```

No Oracle SQL, os dois conjuntos participantes do processo de união são definidos dinamicamente através de dois (ou mais) comandos SELECTs unidos por UNION ou UNION ALL.

Para que seja possível a realização do processo, algumas regras devem ser respeitadas:

- Todos os SELECTs envolvidos devem possuir o mesmo número de colunas.
- As colunas correspondentes em cada um dos comandos devem ser do mesmo tipo.
- A cláusula ORDER BY só se aplica ao resultado geral da união e deve utilizar indicação posicional em vez de expressões.
- As demais cláusulas que compõem um comando SELECT são tratadas individualmente nos comandos a que se aplicam.
- A cláusula UNION elimina do resultado todas as linhas duplicadas. Esta operação pode realizar SORT para garantir a retirada das duplicadas.
- A cláusula UNION ALL apresenta no resultado todas as linhas produzidas no processo de união, independente de serem duplicadas ou não.
- Todos os operadores (UNION, INTERSECT, MINUS) possuem igual precedência.

Exemplo 1: O operador UNION não inclui linhas duplicadas.

```
SELECT nome, cidade, estado  
FROM   tclientes  
WHERE  estado = 'RS'  
UNION  
SELECT nome, cidade, estado  
FROM   tclientes  
WHERE  estado LIKE '%R%'  
ORDER BY 2,3;
```

Operadores SET

RANK	NOME	RANK	CIDADE	RANK	ESTADO
	Alfredo Fonseca		Porto Alegre		RS
	José Batista		Porto Alegre		RS
	Ramiro Antunes		Rio de Janeiro		RJ
	João Medeiros		Rio de Janeiro		RJ
	Nadia Velasques		Rio de Janeiro		RJ
	Ana Maria Prado		Rio de Janeiro		RJ

Note que no SELECT anterior a primeira consulta retorna os clientes do estado igual a 'RS' e a segunda retorna os clientes de 'RS' e 'RJ'. As linhas duplicadas não serão incluídas.

Exemplo 2: O operador UNION ALL inclui as linhas duplicadas

```
SELECT nome, cidade, estado
FROM tclientes
WHERE estado = 'RS'
UNION ALL
SELECT nome, cidade, estado
FROM tclientes
WHERE estado LIKE '%R%'
ORDER BY 2,3;
```

RANK	NOME	RANK	CIDADE	RANK	ESTADO
	José Batista		Porto Alegre		RS
	Alfredo Fonseca		Porto Alegre		RS
	Alfredo Fonseca		Porto Alegre		RS
	José Batista		Porto Alegre		RS
	Nadia Velasques		Rio de Janeiro		RJ
	João Medeiros		Rio de Janeiro		RJ
	Ana Maria Prado		Rio de Janeiro		RJ
	Ramiro Antunes		Rio de Janeiro		RJ

Note que no SELECT anterior a primeira consulta retorna os clientes do estado igual a 'RS' e a segunda retorna os clientes de 'RS' e 'RJ'. As linhas duplicadas serão incluídas.

Utilizando vários operadores SET

Precedência

Em um comando SELECT composto de diversos operadores (UNION, INTERSECT, etc.), o comando é executado da esquerda para a direita. Podemos, neste caso, utilizar parênteses para alterar a ordem de execução.

Exemplo 1:

```
SELECT id, dt_compra, desconto, total, 'UNION 1' QUERY
FROM   tcontratos
WHERE  dt_compra = '10/01/05'
UNION  ALL
SELECT id, dt_compra, desconto, total, 'UNION 2' QUERY
FROM   tcontratos
WHERE  desconto IS NOT NULL
UNION
SELECT id, dt_compra, desconto, total, 'UNION 3' QUERY
FROM   tcontratos
WHERE  total > 4000
ORDER  BY 5;
```

R	ID	R	DT_COMPRA	R	DESCONTO	R	TOTAL	R	QUERY
1	1018	1	10/01/05	1	(null)	1	1912	1	UNION 1
2	1017	1	10/01/05	1	(null)	1	3600	1	UNION 1
3	1003	2	04/01/05	2	985	2	1995	2	UNION 2
4	1005	2	05/01/05	2	191	2	3191	2	UNION 2
5	1007	2	06/01/05	2	50	2	1000	2	UNION 2
6	1008	2	06/01/05	2	700	2	6218	2	UNION 2
7	1019	2	11/01/05	2	60	2	600	2	UNION 2
8	1000	2	03/01/05	2	360	2	3600	2	UNION 2
9	1001	2	04/01/05	2	225	2	4500	2	UNION 2
10	1016	2	09/01/05	2	960	2	15960	2	UNION 2

Neste último exemplo de união, foram somados três conjuntos. No primeiro conjunto, somente as linhas com data de compra igual a '10/01/05'. No segundo conjunto, apenas as linhas com descontos válidos. Com estas restrições não houve necessidade de utilizarmos UNION, uma vez que não haveria linhas duplicadas, tornando a operação mais eficiente, não havendo necessidade de SORT.

Este resultado foi unido ao terceiro conjunto, que tinha como restrição TOTAL > R\$4000, podendo trazer linhas já presentes no conjunto resultante anterior. Desta forma escolhemos utilizar o operador UNION.

Execute o comando do último exemplo e observe que as linhas estão duplicadas (linhas sublinhadas). Isto ocorre porque a avaliação de duplicidade não se dá em relação a

uma coluna em particular e sim em relação a toda a linha. Utilizamos constantes para modificar o resultado das linhas a serem unidas: a constante 'UNION 1' substituiu a coluna QUERY (isto foi possível pois ambas são alfanuméricas, apesar de diferirem em tamanho), a constante 'UNION 2' e a constante 'UNION 3', fazendo com que as linhas provenientes do terceiro SELECT fossem apresentadas integralmente.

Exemplo 2:

```
SELECT id, dt_compra, desconto, total
FROM   tcontratos
WHERE  dt_compra = '10/01/05'
UNION  ALL
SELECT id, dt_compra, desconto, total
FROM   tcontratos
WHERE  desconto IS NOT NULL
UNION
SELECT id, dt_compra, desconto, total
FROM   tcontratos
WHERE  total > 4000
ORDER  BY 1;
```

ID	DT_COMPRA	DESCONTO	TOTAL
1000	03/01/05	360	3600
1001	04/01/05	225	4500
1003	04/01/05	985	1995
1004	05/01/05	(null)	9000
1005	05/01/05	191	3191
1007	06/01/05	50	1000
1008	06/01/05	700	6218
1011	07/01/05	1500	3000
1012	08/01/05	60	600
1013	08/01/05	260	2600
1015	09/01/05	(null)	4200
1016	09/01/05	960	15960

Interseção - INTERSECT

A operação de interseção restringe o conjunto resultante às tuplas presentes em todos os conjuntos participantes da operação. Considerando-se a existência de duas tabelas A e B, as linhas resultantes da interseção seriam representadas por L A B. A interseção só pode ser efetuada entre relações união compatíveis.

INTERSECT . . .

No SQL do Oracle, os dois conjuntos participantes do processo de interseção são definidos dinamicamente através de dois (ou mais) comandos SELECTs unidos por INTERSECT.

As mesmas regras aplicáveis à união são válidas na interseção:





- Todos os SELECTs envolvidos devem possuir o mesmo número de colunas.
- As colunas correspondentes em cada um dos comandos devem ser do mesmo tipo.
- A cláusula ORDER BY só se aplica ao resultado geral da interseção e deve utilizar indicação posicional em vez de expressões.
- As demais cláusulas que compõem um comando SELECT são tratadas individualmente nos comandos a que se aplicam.
- A cláusula INTERSECT elimina do resultado todas as linhas duplicadas. Esta operação pode realizar sort para garantir a retirada das duplicadas.
- Todos os operadores (UNION, INTERSECT, MINUS) possuem igual precedência.

Em um comando SELECT composto de diversos operadores (UNION, INTERSECT, etc.), o comando é executado da esquerda para a direita. Podemos, neste caso, utilizar parênteses para alterar a ordem de execução.

Exemplo:

```
SELECT id, dt_compra, desconto, total
FROM   tcontratos
WHERE  desconto IS NOT NULL
INTERSECT
SELECT id, dt_compra, desconto, total
FROM   tcontratos
WHERE  total > 4000
ORDER  BY 1;
```

Operadores SET

 ID	 DT_COMPRA	 DESCONTO	 TOTAL
1001	04/01/05	225	4500
1008	06/01/05	700	6218
1016	09/01/05	960	15960

Diferença - MINUS

A operação de diferença efetua uma subtração de conjuntos eliminando as duplicidades. Considerando a existência de duas tabelas A e B, o resultado L é uma relação que contém as linhas de A que não estão presentes em B. As linhas da diferença seriam representadas por $L = A - B$. A diferença só pode ser efetuada entre relações união-compatíveis.

No Oracle SQL, os dois conjuntos participantes do processo de diferença são definidos dinamicamente através de dois (ou mais) comandos SELECTs unidos pelo operador MINUS.

MINUS ...

As regras já apresentadas na UNION e INTERSECT devem ser obedecidas na diferença.

Exemplo 1:

```
SELECT id, dt_compra, desconto, total
FROM   tcontratos
WHERE  dt_compra = '10/01/05'
MINUS
SELECT id, dt_compra, desconto, total
FROM   tcontratos
WHERE  desconto IS NOT NULL;
```

ID	DT_COMPRA	DESCONTO	TOTAL
1017	10/01/05	(null)	3600
1018	10/01/05	(null)	1912

Exemplo 2:

```
SELECT id, dt_compra, desconto, total
FROM   tcontratos
WHERE  desconto IS NOT NULL
MINUS
SELECT id, dt_compra, desconto, total
FROM   tcontratos
WHERE  total > 4000
ORDER BY 1;
```

Operadores SET

ID	DT_COMPRA	DESCONTO	TOTAL
1000	03/01/05	360	3600
1003	04/01/05	985	1995
1005	05/01/05	191	3191
1007	06/01/05	50	1000
1011	07/01/05	1500	3000
1012	08/01/05	60	600
1013	08/01/05	260	2600
1019	11/01/05	60	600

Exercícios – 9

1. No SQL Developer utilize a sua conexão e conecte-se ao banco de dados do curso. Crie uma expressão UNION que represente a união dos totais de cada contrato e das somas dos itens de contrato. No primeiro SELECT selecione o código do contrato, o valor total de todos os contratos e a string fixa definida por 'CONTRATOS'. No segundo SELECT selecione o código do contrato, todas as somas do total de cada item deste contrato e a string fixa 'ITENS'. Ordene pela terceira coluna e depois pela primeira. Defina os alias nos dois SELECTs como ID, TOTAL e STRING. Execute a consulta.
2. Crie um script para a operação de subtração (MINUS) entre dois SELECTs e a intersecção (INTERSECT) com um terceiro SELECT. No primeiro SELECT selecione ID, DT_COMPRA e TOTAL de todos os contratos. No segundo SELECT selecione ID, DT_COMPRA e TOTAL de todos os contratos sem desconto (NULO ou ZERO). Subtraia o primeiro pelo segundo. No terceiro SELECT selecione ID, DT_COMPRA e TOTAL dos contratos realizados com clientes de SP. Execute a consulta.

Espaço para anotações

10. Manipulando Dados

Objetivos

- Descrever cada comando DML;
- Inserir linhas em uma tabela;
- Atualizar linhas de uma tabela;
- Remover linhas de uma tabela;
- Controlar transações.

Linguagem de Manipulação de Dados

Linguagem de manipulação de dados (DML) é uma parte essencial do SQL. Quando você necessita adicionar, atualizar ou apagar dados no banco de dados, você executa um comando DML. Um conjunto de comandos DML forma uma unidade lógica de trabalho chamada de transação.

Considere um banco de dados bancário. Quando um cliente do banco transfere dinheiro de uma poupança para uma conta corrente, a transação poderia consistir de três operações separadas: diminuir da poupança, aumentar na conta corrente e registrar a transação no diário de transações. O Servidor Oracle deve garantir que todos os três comandos SQL sejam executados para manter as contas equilibradas. Quando algo impede um dos comandos da transação de executar, devem ser desfeitas as mudanças causadas pelos outros comandos da transação.

Comando INSERT

Você pode adicionar linhas novas para uma tabela executando um comando INSERT.

```
INSERT INTO tabela [(coluna [, coluna...])]  
VALUES (valor [, valor...]);
```

Sintaxe:

tabela: é o nome da tabela.

coluna: é o nome da coluna da tabela que receberá os valores.

valor: é o valor correspondente para a coluna.

Nota: Este comando com a cláusula VALUES adiciona apenas uma linha de cada vez para a tabela.

Inserindo Novas Linhas

```
INSERT INTO tdescontos(classe, base_inferior, base_superior)
VALUES ('F', 50000, 99999);
```

Uma vez que você pode inserir uma linha nova que contenha valores para cada coluna da tabela, a lista de colunas não é obrigatória na cláusula INSERT. Porém, se você não utilizar a lista de colunas, os valores devem ser listados de acordo com a ordem default das colunas na tabela.

Por questões de clareza, utilize a lista de colunas na cláusula INSERT. Coloque valores caractere e data entre aspas simples; não inclua valores numéricos entre aspas.

Inserindo Linhas com Valores Nulos

- Método implícito:

```
INSERT INTO tdescontos(classe, base_inferior)
VALUES ('F', 50000);
```

```
INSERT INTO tdescontos
VALUES('F', 50000, NULL);
```

- Método explícito:

ou

```
INSERT INTO tdescontos
VALUES('F', 50000, '');
```

Método	Descrição
Implícito	Omita a coluna da lista de colunas
Explícito	Especifique a palavra chave NULL na lista da cláusula VALUES. Especifique uma string vazia (' ') na lista da cláusula VALUES

Tabela 8-1: Métodos de inserção

Tenha certeza que a coluna de destino permita valores nulos verificando o status da coluna Nulo? do comando DESCRIBE do SQL*Plus.

O Servidor Oracle automaticamente verifica todos os tipos de dados, intervalos de valores e regras de integridade de dados. Qualquer coluna que não é listada explicitamente recebe um valor nulo na linha nova.

Inserindo Valores Especiais

Você pode utilizar pseudocolunas, expressões e funções para entrar valores especiais em sua tabela.

```
INSERT INTO tcontratos(id, dt_compra, status, tclientes_id,
desconto, total)
VALUES (1021, SYSDATE, 'A', 150, NULL, 3000);
```

O exemplo acima armazena informações para o contrato 1021 na tabela TCONTRATOS. Ele armazena a data e hora atual na coluna DT_COMPRA, utilizando a função SYSDATE.

Você também pode utilizar a função USER quando inserir linhas em uma tabela. A função USER armazena o nome do usuário atual.

Confirmando a Inserção:

```
SELECT *
FROM   tcontratos
WHERE  id = 1021;
```

ID	DT_COMPRA	STATUS	TCLIENTES_ID	DESCONTO	TOTAL
1021	12/06/09	A	150	(null)	3000

Inserindo Valores de Data Específicos

- Adicione um novo contrato:

```
INSERT INTO tcontratos  
VALUES (1022,TO_DATE('16/01/2009','DD/MM/YYYY'),'A',200,100,1000);
```

O formato DD/MM/YY é normalmente utilizado para inserir um valor de data. Com este formato. Uma vez que a data também possui informação de hora, a hora default é meia-noite (00:00:00).

Se uma data precisa ser entrada em outro século ou com uma hora específica, utilize a função TO_DATE.

O exemplo acima armazena informações para o contrato 1022 na tabela TCONTRATOS. O valor da coluna DT_COMPRA fica sendo 16 de Janeiro de 2009 (00:00:00 hora).

Inserindo Valores Utilizando Variáveis de Substituição

```
INSERT INTO tdescontos(classe, base_inferior, base_superior)
VALUES ('&classe', &base_inferior, &base_superior);
```

Você pode produzir um comando INSERT que permite ao usuário adicionar valores interativamente utilizando variáveis de substituição do SQL*Plus.

O exemplo acima armazena informações para uma classe de desconto na tabela TDESCONTOS. Ele solicita ao usuário a classe de desconto, o valor da base inferior e o valor da base superior.

Para valores caractere e data, o símbolo (&) e o nome da variável deve ficar entre aspas simples.

Criando um Scripts SQL com Prompts Customizados

```
ACCEPT classe PROMPT 'Informe o valor para a classe de desconto:'  
ACCEPT base_inferior PROMPT 'Informe o valor para a base inferior:'  
ACCEPT base_superior PROMPT 'Informe o valor para a base superior:'  
INSERT INTO tdescontos(classe, base_inferior, base_superior)  
VALUES ('&classe', &base_inferior, &base_superior);
```

Você pode salvar seu comando com variáveis de substituição para um arquivo e então executá-lo. Cada vez que você executa o comando, ele solicitará novos valores. Customize os prompts utilizando o comando ACCEPT do SQLPlus.

O exemplo acima armazena informações para uma classe de desconto na tabela TDESCONTOS. Ele solicita ao usuário a classe de desconto, a base inferior de desconto e a base superior de desconto utilizando mensagens customizadas.

Não prefixe parâmetros de substituição com o símbolo (&) quando referenciá-los no comando ACCEPT. Utilize um hífen (-) para continuar um comando do SQLPlus na próxima linha.

INSERT utilizando uma sub-consulta

Você pode inserir linhas em uma tabela a partir do resultado de uma sub-consulta.

- Não utilize a cláusula VALUES.
- O número de colunas ou expressões na cláusula INSERT deve ser igual ao número de colunas da sub-consulta.

Sintaxe:

```
INSERT into nome_tabela1
(coluna1, coluna1 [, coluna..] )
SELECT coluna1, coluna2 [, coluna..]
FROM nome_tabela2
WHERE expressão
```

Exemplo:

```
INSERT into clientes_rs
(id, nome, dt_nascimento, endereco, cidade, estado, cep,
telefone, comentarios)
SELECT id, nome, dt_nascimento, endereco, cidade, estado,
cep, telefone, comentarios
FROM tclientes
WHERE estado = 'RS';
```

Comando UPDATE

Você pode modificar linhas existentes utilizando o comando UPDATE.

```
UPDATE tabela
SET      coluna = valor | expressão [, coluna =
valor | expressão]
[WHERE condição];
```

Sintaxe:

tabela: é o nome da tabela.

coluna: é o nome da coluna a alterar.

valor: é o valor, expressão correspondente ou Sub-consulta a ser atribuída para a coluna.

condição: identifica as linhas a serem atualizadas; é composto de nomes de colunas, expressões, constantes, Sub-consultas e operadores de comparação.

Confirme a operação de atualização examinando a tabela e exibindo as linhas atualizadas.

Nota: Em geral, utilize a chave primária para identificar uma única linha. Utilizar outras colunas pode causar a atualização indesejada de várias linhas. Por exemplo, identificar uma única linha da tabela TCONTRATOS através da data de compra é perigoso porque mais de um contrato pode ter a mesma data de compra.

Alterando Linhas em uma Tabela

Linhas específicas são modificadas quando você utiliza uma cláusula WHERE.

```
UPDATE tcontratos  
SET     desconto = total * 0.5  
WHERE   id = 1000;
```

Todas as linhas da tabela são modificadas se você omitir a cláusula WHERE.

```
UPDATE tcontratos  
SET     desconto = total * 0.5;
```

O comando UPDATE modifica linhas específicas, se a cláusula WHERE for especificada. O exemplo acima altera o valor do desconto de todas as linhas da tabela TCONTRATOS para 50% do valor do total de cada contrato.

UPDATE utilizando uma sub-consulta

Você pode atualizar colunas de uma tabela a partir do resultado de uma sub-consulta.

- A sub-consulta deve ficar entre parênteses e retornar uma e somente uma linha

Sintaxe:

```
UPDATE nome_tabela1
SET coluna1 = (SELECT coluna1
               FROM nome_tabela1),
    coluna2 = (SELECT coluna2
               FROM nome_tabela2)
[, colunaN = (SELECT colunaN
               FROM nome_tabelaN)..]
WHERE expressão;
```

Exemplo 1:

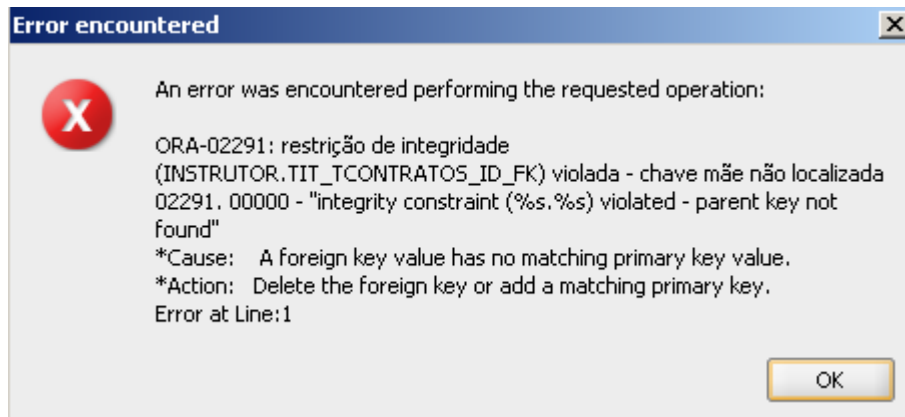
```
UPDATE media_contratos_clientes me
SET    me.total = (SELECT AVG(col.total)
                  FROM    tcontratos col
                  WHERE me.tclientes_id =
col.tclientes_id),
      me.desconto = (SELECT AVG(co2.desconto)
                    FROM tcontratos co2
                    WHERE me.tclientes_id =
co2.tclientes_id);
```

Exemplo 2

```
UPDATE media_contratos_clientes me
SET  (me.total, me.desconto) =
     (SELECT AVG(col.total), AVG(col.desconto)
      FROM    tcontratos col
      WHERE me.tclientes_id =
col.tclientes_id);
```

Atualizando Linhas: Erro de Constraint de Integridade

```
UPDATE titens  
SET tcontratos_id = 1  
WHERE tcontratos_id = 1000;
```



Se você tentar atualizar um registro com um valor que invalide uma constraint de integridade, você receberá um erro.

No exemplo acima, o contrato de id igual a 1 não existe na tabela pai (TCONTRATOS), portanto você recebe o erro "parent key not found", ORA-02291.

Nota: Constraints de integridade asseguram que os dados sigam um conjunto pré-determinado de regras. Um capítulo subsequente apresentará mais detalhes sobre as constraints de integridade.

Comando DELETE

Você pode remover linhas utilizando o comando DELETE.

```
DELETE [FROM] tabela  
[WHERE condição];
```

Sintaxe:

tabela: é o nome da tabela.

condição : condição que seleciona as linhas a serem removidas; é composta de nomes de colunas, expressões, constantes, Sub-consultas e operadores de comparação.

Removendo Linhas de uma Tabela

Linhas específicas são removidas quando você utiliza a cláusula WHERE.

```
DELETE FROM tdescontos  
WHERE classe = 'A';
```

Todas as linhas da tabela são removidas se você omitir a cláusula WHERE.

```
DELETE FROM tdescontos;
```

Você pode remover linhas específicas utilizando a cláusula WHERE no comando DELETE. O exemplo acima remove os descontos TDESCONTOS. Você pode confirmar a operação de exclusão tentando exibir as linhas removidas utilizando o comando SELECT.

```
SELECT *  
FROM tdescontos;
```

DELETE utilizando uma sub-consulta

Você pode remover linhas de uma tabela a partir do resultado de uma sub-consulta.

- A sub-consulta deve ficar entre parênteses e retornar uma e somente uma linha

Sintaxe:

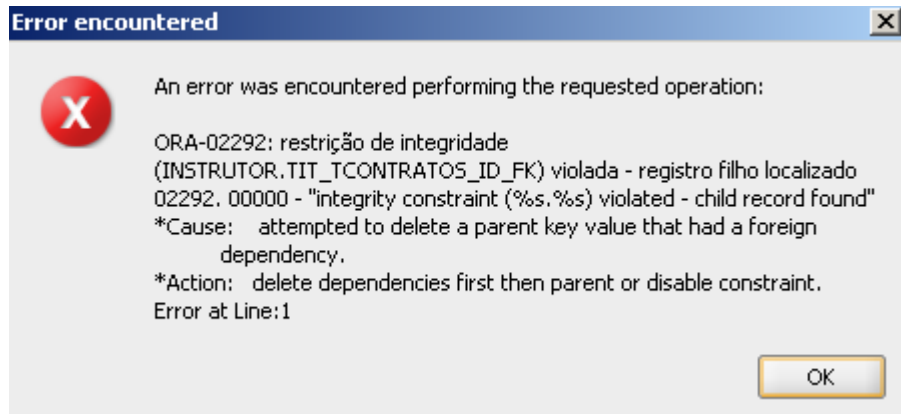
```
DELETE FROM nome_tabela1
WHERE coluna1 | expressão operador
      (SELECT coluna1
        FROM nome_tabela1);
```

Exemplo:

```
DELETE FROM media_contratos_clientes me
WHERE me.total > (SELECT AVG(col.total)
                  FROM tcontratos col);
```


Removendo Linhas: Erro de Constraint de Integridade

```
DELETE FROM tcontratos  
WHERE id = 1000;
```



Se você tentar remover um registro com um valor que invalide uma constraint de integridade, você receberá um erro.

O exemplo acima tenta remover os contratos da tabela TCONTRATOS com ID = 1000 mas resulta em um erro porque o contrato com ID = 1000 possui itens na tabela TITENS, e este contrato é utilizado como chave estrangeira definida sobre a coluna titens.tcontratos_id. Se o registro pai que você tentou apagar possuir registros filhos, então você recebe a mensagem de erro "child record found", ORA-02292.

Transações de Banco de Dados

O Servidor Oracle garante a consistência dos dados baseado em transações. Transações fornecem maior flexibilidade e controle para a modificação dos dados, assegurando a consistência dos mesmos no caso de falha do processo do usuário ou falha do sistema.

Transações consistem em comandos DML que compõem uma mudança consistente dos dados. Por exemplo, uma transferência de fundos entre duas contas deve incluir o débito para uma conta e o crédito para outra conta no mesmo valor. Ambas as ações devem falhar ou devem ter sucesso juntas. O crédito não pode ser efetuado sem o correspondente débito.

Tipos de Transações

Tipo	Descrição
Data manipulation language (DML)	Consiste de um número de comandos DML que o Servidor Oracle trata como uma única entidade ou unidade lógica de trabalho
Data definition language (DDL)	Consiste de um único comando DDL
Data control language (DCL)	Consiste de um único comando DCL

Tabela 8-2: Tipos de transações

Quando uma Transação Inicia?

- Uma transação inicia quando o primeiro comando SQL DML é executado.

Quando uma Transação Termina?

Quando um dos seguintes eventos acontece:

- Um comando COMMIT ou ROLLBACK é executado
- Um comando DDL, como CREATE, é executado
- Um comando DCL é executado
- O usuário encerra o SQL*Plus
- O servidor de banco de dados Oracle cai.

Nota: Depois que uma transação termina, o próximo comando SQL executado iniciará a próxima transação automaticamente.

Um comando DDL ou DCL é automaticamente confirmado (commit) e portanto implicitamente termina a transação.

Vantagens do COMMIT e ROLLBACK

- Garantem a consistência dos dados;
- Visualização dos dados modificados antes de tornar as modificações permanentes;
- Agrupam logicamente operações relacionadas.

Controlando Transações

Você pode controlar a lógica das transações utilizando os comandos COMMIT, SAVEPOINT e ROLLBACK.

Comando	Descrição
COMMIT	Termina a transação corrente tornando todas as modificações pendentes permanentes
SAVEPOINT <i>nome</i>	Coloca uma marca dentro da transação corrente
ROLLBACK [TO <i>SAVEPOINT nome</i>]	ROLLBACK termina a transação corrente desfazendo todas as modificações pendentes. ROLLBACK TO <i>SAVEPOINT nome</i> desfaz todas as modificações após a marca de <i>savepoint</i>

Tabela 8-3: Comandos de controle de transação

Nota: SAVEPOINT não faz parte do padrão SQL ANSI.

Processamento Implícito de Transações

Situação	Circunstâncias
Commit automático	Comando DDL ou comando DCL é executado Saída normal do SQL*Plus, sem explicitamente executar um COMMIT ou ROLLBACK
Rollback automático	Saída anormal do SQL*Plus ou falha do sistema

Tabela 8-4: Processamento implícito de transações

Nota: Um terceiro comando está disponível no SQLPlus. O comando do SQLPlus AUTOCOMMIT pode ser alternado para ON ou OFF. Se for setado para ON, cada comando DML individual sofre commit assim que for executado. Você não pode desfazer as mudanças. Se for setado para OFF, o COMMIT pode ser executado explicitamente. Também, o COMMIT é efetuado quando um comando DDL é executado ou quando você encerra o SQLPlus.

Falhas do Sistema

Quando uma transação é interrompida por uma falha de sistema no servidor Oracle, queda na conexão ou o usuário desconectou de forma anormal a transação inteira é desfeita automaticamente. Isto previne o erro de causar mudanças não desejadas para os dados e retorna as tabelas para o seu estado no momento do último commit. Desta forma, o SQL protege a integridade das tabelas.

Situação dos Dados Antes do COMMIT ou ROLLBACK

Toda mudança dos dados feita durante a transação é temporária até a transação sofrer commit.

Situação dos Dados Antes do COMMIT ou ROLLBACK

- Operações de manipulação de dados inicialmente afetam o buffer do banco de dados; portanto, o estado anterior dos dados pode ser recuperado.
- O usuário atual pode visualizar os resultados das operações de manipulação de dados examinando as tabelas.
- Outras sessões não podem visualizar os resultados das operações de manipulação de dados efetuadas pelo usuário atual. O Servidor Oracle fornece leitura consistente para assegurar que cada usuário visualize os dados como ficaram após o último commit.
- As linhas afetadas são bloqueadas; outras sessões não podem modificar os dados destas linhas.

Situação dos Dados Após o COMMIT

Efetive as mudanças pendentes utilizando o comando COMMIT. Após o COMMIT:

- Modificações para os dados são efetivadas no banco de dados.
- O estado anterior dos dados fica permanentemente perdido.
- Todos os usuários podem visualizar os resultados da transação.
- Os locks nas linhas afetadas são liberados; as linhas ficam disponíveis para outras sessões executarem novas mudanças nos dados.
- Todos o savepoints são eliminados.

Efetivando os Dados

- Faça as alterações:

```
UPDATE tcontratos  
SET desconto = total * 0.5  
WHERE id = 1000;
```

- Efetive as alterações:

```
COMMIT;
```

O exemplo acima atualiza a tabela TCONTRATOS alterando o valor de desconto para 50% do valor total do contrato 1000. Depois ele torna a mudança permanente executando o comando COMMIT.

Situação dos Dados Após o ROLLBACK

```
DELETE FROM titens;  
ROLLBACK;
```

Descarte todas as mudanças pendentes utilizando o comando ROLLBACK. Após o ROLLBACK:

- Modificações para os dados são desfeitas.
- O estado anterior dos dados é restaurado.
- Os locks nas linhas afetadas são liberados.

Exemplo:

Ao tentar remover um registro da tabela TITENS, você pode acidentalmente apagar toda a tabela. Você pode corrigir o engano, e então executar o comando novamente de forma correta, tornando as mudanças permanentes.

```
DELETE FROM titens;  
ROLLBACK;  
DELETE FROM titens  
WHERE tcontratos_id = 1000;  
COMMIT;
```

Utilizando Savepoints

Desfazendo as Alterações Até uma Marca.

```
UPDATE tcursos
SET    preco = 2000
WHERE  id = 50;
SAVEPOINT ok;
UPDATE tcursos
SET    carga_horaria = 30
WHERE  id = 50;
ROLLBACK TO SAVEPOINT ok;
COMMIT;
```

Você pode criar uma marca dentro da transação corrente utilizando o comando **SAVEPOINT**. A transação pode ser dividida então em seções menores. Você pode descartar as mudanças pendentes até aquela marca utilizando o comando **ROLLBACK TO SAVEPOINT**.

Se você criar um segundo savepoint com o mesmo nome de um savepoint anterior, o savepoint mais anterior é removido.

Rollback ao Nível de Comando

Parte de uma transação pode ser descartada por um rollback implícito se um erro de execução de comando for detectado. Se um comando DML falhar durante sua execução em uma transação, seu efeito será descartado por um rollback ao nível de comando, mas as mudanças feitas pelos comandos DML anteriores na transação não serão descartadas. Eles podem ser confirmados ou descartados explicitamente pelo usuário.

O Oracle executa um COMMIT implícito antes e depois de qualquer comando de definição de dados (DDL). Portanto, se seu comando DDL não executar prosperamente, você não poderá desfazer as mudanças anteriores porque o servidor já emitiu um commit.

Termine suas transações explicitamente executando um comando COMMIT ou ROLLBACK.

Leitura Consistente

- Usuários de banco de dados efetuam dois tipos de acesso ao banco de dados:
- Operações de leitura (comando SELECT)
- Operações de escrita (comandos INSERT, UPDATE e DELETE)
- Você necessita de leitura consistente para que o seguinte aconteça:
- A leitura e gravação do banco de dados são garantidas com uma visão consistente dos dados.
- Leituras não visualizam dados que ainda estão em processo de atualização.
- Escritas para o banco de dados garantem que as mudanças são efetuadas de uma forma consistente.
- Mudanças feitas por um usuário não conflitam com mudanças que outro usuário está fazendo.
- O propósito da leitura consistente é assegurar que cada usuário visualize os dados como eles ficaram antes do último commit, antes da operação DML começar.

Implementação de Leitura Consistente

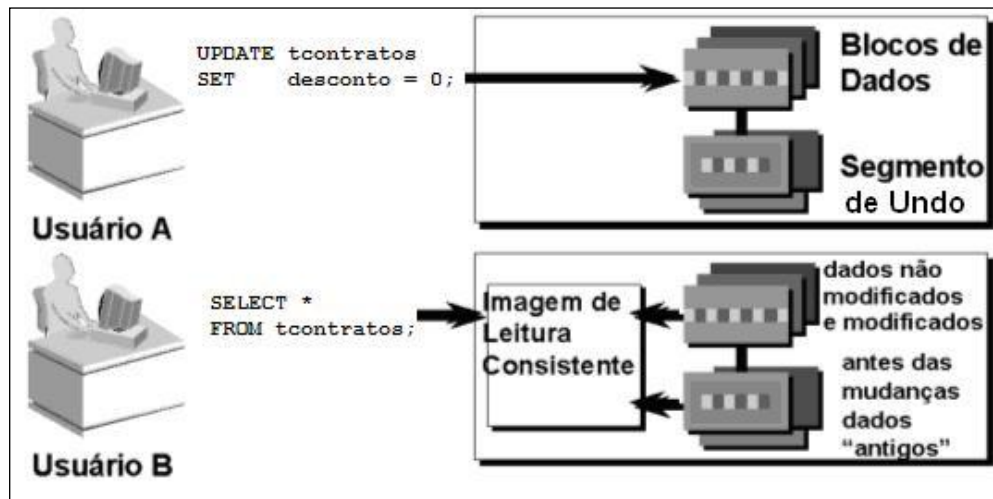


Figura 8-2: Implementação de leitura consistente

Leitura consistente é uma implementação automática. Mantém uma cópia parcial do banco de dados em segmentos de undo.

Quando uma operação de inserção, atualização ou deleção é feita no banco de dados, o Servidor Oracle faz uma cópia dos dados antes deles serem modificados e a armazena em um segmento de undo.

Todas as sessões, exceto o que executou a mudança, ainda visualizam o banco de dados como estava antes do início das mudanças, eles visualizam um "snapshot" dos dados a partir dos segmentos de undo.

Antes das mudanças sofrerem commit no banco de dados, somente a sessão que está modificando os dados visualiza o banco de dados com as alterações, qualquer outra sessão visualiza um snapshot no segmento de undo. Isto garante que os leitores dos dados leiam dados consistentes que não estão sofrendo mudanças atualmente.

Quando um comando DML sofre commit, as mudanças feitas ao banco de dados tornam-se visíveis para qualquer usuário executando um comando SELECT. O espaço ocupado pelos dados "antigos" no arquivo do segmento de undo é liberado para ser reutilizado.

Se a transação sofrer rollback, as mudanças são desfeitas.

- O original, versão antiga, dos dados no segmento de undo é escrito de volta à tabela.
- Todos os usuários visualizam o banco de dados como estava antes da transação iniciar.

Lock

O que São Locks?

Locks são mecanismos que previnem interação destrutiva entre transações que acessam o mesmo recurso, ou um objeto de usuário (como tabelas ou linhas) ou objetos de sistema não visíveis aos usuários (como estruturas de dados compartilhados e linhas do dicionário de dados).

Como o Oracle Efetua o Lock dos Dados

Lock em um banco de dados Oracle é totalmente automático e não requer nenhuma ação por parte do usuário. Lock implícito ocorre para todos os comandos SQL exceto SELECT. O mecanismo padrão de lock Oracle automaticamente utiliza o mais baixo nível aplicável de restrição, fornecendo um nível maior de concorrência, também provendo o máximo em integridade de dados. O Oracle também permite ao usuário efetuar locks dos dados manualmente.

SELECT FOR UPDATE

- Um select com a cláusula FOR UPDATE efetuará o bloqueio (Lock) nas linhas recuperadas pelo comando SELECT.
- As linhas bloqueadas (Locked) serão liberadas somente quando a você emitir um comando COMMIT ou ROLLBACK.
- Se o comando SELECT tentar bloquear (lock) uma linha que já esta bloqueada por outra transação de outra sessão, então o banco de dados Oracle espera até que a linha fique disponível, e então recupera a linha e efetua o bloqueio (lock) da mesma.

Exemplo:

```
SELECT id, nome, preco
FROM   tcursos
WHERE  id = 50
FOR    UPDATE;
```

SELECT FOR UPDATE utilizando mais de uma tabela

- Você pode utilizar a cláusula FOR UPDATE no comando SELECT sobre múltiplas tabelas.

No exemplo a seguir as linhas recuperadas da tabela TCONTRATOS e TCLIENTES serão bloqueadas (locked).

```
SELECT co.id, co.total, co.desconto,  
       cl.nome  
FROM   tcontratos co, tclientes cl  
WHERE  (co.tclientes_id = cl.id) AND  
       (co.id = 1001)  
FOR    UPDATE;
```

- Você pode utilizar a cláusula FOR UPDATE OF *nome_coluna1*, *nome_coluna2* para qualificar a(s) que você deseja alterar, desta maneira somente as linhas recuperadas da tabela que contem a(s) coluna(s) especifica serão bloqueadas (locked).

No exemplo a seguir somente as linhas recuperadas da tabela TCONTRATOS serão bloqueadas (locked).

```
SELECT co.id, co.total, co.desconto,  
       cl.nome  
FROM   tcontratos co, tclientes cl  
WHERE  (co.tclientes_id = cl.id) AND  
       (co.id = 1001)  
FOR    UPDATE of co.total, co.desconto;
```


Exercícios – 10

1. No SQL Developer utilize a sua conexão e conecte-se ao banco de dados do curso. Descreva a estrutura da tabela TCLIENTES_VIP para identificar os nomes das colunas.
2. Adicione a primeira linha de dados na tabela TCLIENTES_VIP a partir do exemplo de dados a seguir.

ID	SOBRENOME	NOME	CLIENTEID	CREDITO
1	Tapes	Carlos	ctapes	795

3. Adicione a segunda linha na tabela TCLIENTES_VIP a partir do exemplo de dados a seguir.

ID	SOBRENOME	NOME	CLIENTEID	CREDITO
2	Moura	Ana	amoura	860

4. Visualize suas inserções para a tabela TCLIENTES_VIP.
5. Adicione mais três linhas na tabela TCLIENTES_VIP a partir do exemplo de dados a seguir.

ID	SOBRENOME	NOME	CLIENTEID	CREDITO
3	Pinheiro	Viviane	vpinheir	1100
4	Dutra	Manuel	mdutra	750
5	Silva	Cesar	csilva	1550

6. Visualize suas inserções para a tabela TCLIENTES_VIP.
7. Torne as inserções permanentes.
8. Modifique o sobrenome do cliente com ID = 3 para "Souza".
9. Modifique o crédito para 1000 para todos os clientes com o crédito menor que 900.
10. Visualize suas modificações para a tabela TCLIENTES_VIP.
11. Remova o cliente "Ana Moura" da tabela TCLIENTES_VIP.
12. Visualize suas modificações para a tabela.
13. Efetive todas as modificações pendentes.

COMMIT ;

14. Adicione uma linha para a tabela TCLIENTES_VIP com os valores id = 6, nome = 'Roberto', sobrenome = 'Pires', clienteid = rpires e credito = 2000.
15. Visualize sua inserção para a tabela.

16. Marque um ponto intermediário no processamento da transação.
17. Remova todas as linhas da tabela TCLIENTES_VIP.
18. Visualize que a tabela TCLIENTES_VIP está vazia.
19. Descarte a mais recente operação de DELETE sem descartar a operação de INSERT anterior.
20. Visualize a tabela TCLIENTES_VIP.
21. Efetive a transação.

COMMIT ;

Espaço para anotações

11. Criando e Gerenciando Tabelas

Objetivos

- Descrever os principais objetos do banco de dados;
- Criar tabelas;
- Descrever os tipos de dados que podem ser utilizados na definição de colunas;
- Alterar a definição de tabelas;
- Remover, renomear e truncar tabelas.

Objetos do Banco de Dados

Um banco de dados Oracle pode conter várias estruturas de dados. Cada estrutura deve ser pensada no design do banco de dados de forma que possa ser criada durante a fase de construção e desenvolvimento do mesmo.

Objeto	Descrição
Tabela (Table)	Unidade básica de armazenamento. Composta por linhas e colunas.
Visões (View)	Representação lógica de um subconjunto de dados de uma ou mais tabelas.
Sequencia (Sequence)	Estrutura que gera números sequenciais para chaves primárias.
Índice (Index)	Estrutura que aumenta a performance de consultas.
Sinônimo (Synonym)	Estrutura que especifica um outro nome para um objeto.

Estruturas de Tabelas Oracle

Tabelas podem ser criadas a qualquer momento, até mesmo enquanto os usuários estiverem utilizando o banco de dados.

Você não precisa especificar o tamanho da tabela. O tamanho pode ser definido posteriormente pela quantidade de espaço alocado para o banco de dados como um todo. Porém, é importante calcular quanto espaço utilizará uma tabela com o passar do tempo.

Estruturas de tabelas podem ser modificadas de forma on-line.

Nota: Outros objetos de banco de dados estão disponíveis, mas não serão estudados neste curso.

Convenções de Nomes

Nomeie tabelas e colunas de acordo com o padrão de nomenclatura para qualquer objeto do banco de dados Oracle:

- Nomes de tabela e colunas devem começar com uma letra e podem ter de 1 até 30 caracteres de tamanho.
- Nomes devem conter somente os caracteres A–Z, a–z, 0–9, _ (underscore), \$ e #.
- Nomes não devem possuir o mesmo nome de outro objeto criado no schema do mesmo usuário do Servidor Oracle.
- Nomes não devem ser uma palavra reservada do Oracle.

Diretrizes de Nomenclatura

- Utilize nomes descritivos para tabelas e outros objetos do banco de dados.

Nota: Nomes não fazem distinção entre maiúsculas e minúsculas. Por exemplo, TCLIENTES é tratado da mesma forma que tCLIENTES ou tclienteS.

Comando CREATE TABLE

```
CREATE TABLE [schema.]tabela  
  (coluna tipo_de_dado [DEFAULT expr] [, ..]);
```

Crie tabelas para armazenar dados executando o comando SQL CREATE TABLE. Este comando é um dos comandos da linguagem de definição de dados (DDL), que serão discutidos nos próximos capítulos. Comandos DDL são um subconjunto dos comandos SQL utilizados para criar, modificar ou remover estruturas de banco de dados Oracle. Estes comandos possuem um efeito imediato no banco de dados, e eles também registram informações no dicionário de dados.

Para criar uma tabela, o usuário deve possuir o privilégio CREATE TABLE e uma área de armazenamento na qual criará os objetos. O administrador do banco de dados utiliza comandos da linguagem de controle de dados (DCL), que serão discutidos em um capítulo posterior, para conceder privilégios aos usuários.

Sintaxe:

schema: é igual ao nome do usuário dono do objeto.

tabela: é o nome da tabela.

DEFAULT expr: especifica um valor default se um valor for omitido para a coluna no comando INSERT.

coluna: é o nome da coluna.

Tipo_de_dado: é o tipo de dado e tamanho da coluna.

Opção DEFAULT

Exemplo:

```
... dt_compra DATE DEFAULT SYSDATE, ...
```

- Uma coluna pode receber um valor default através da opção DEFAULT. Esta opção impede que valores nulos entrem nas colunas se uma linha é inserida sem um valor para esta coluna. O valor default pode ser uma literal, uma expressão ou uma função SQL, como SYSDATE e USER.
- O valor não pode ser o nome de outra coluna ou uma pseudocoluna, como NEXTVAL no caso de sequences que veremos adiante. A expressão default deve corresponder ao tipo de dado da coluna.

Criando Tabelas

- Crie a tabela:

```
CREATE TABLE tdescontos
(classe          VARCHAR2 (2) NOT NULL,
base_inferior    NUMBER (7,2) ,
base_superior    NUMBER (7,2));
```

- Confirme a criação da tabela:

```
DESC tdescontos
```

Name	Null	Type
-----	-----	-----
CLASSE	NOT NULL	VARCHAR2 (2)
BASE_INFERIOR		NUMBER (7,2)
BASE_SUPERIOR		NUMBER (7,2)

O exemplo acima cria a tabela TDESCONTOS, com três colunas chamadas CLASSE, BASE_INFERIOR e BASE_SUPERIOR. Confirme a criação da tabela executando o comando DESCRIBE.

Uma vez que o comando de criação de tabelas é do tipo DDL, um commit automático ocorre quando este comando é executado.

Consultando o Dicionário de Dados

Uma vez que o Oracle gerencia todos os seus objetos no próprio banco de dados, é possível para o usuário ter acesso a estas informações. Estas informações estão armazenadas em tabelas no banco e são disponibilizadas para consulta em forma de visões. Estas tabelas e visões compõem o que chamamos de Dicionário de Dados.

Existem três grupos de visões: **USER**, **ALL** e **DBA**. Estes grupos funcionam como uma espécie de filtro e estão acessíveis dependendo das permissões do usuário. As visões **USER_*** mostram objetos do próprio usuário; as visões **ALL_*** mostram objetos do usuário e objetos aos quais o usuário tem acesso (através de GRANT); e as visões **DBA_*** mostram todos os objetos do banco de dados.

O nome das visões é bastante intuitivo e é formado pelo grupo, funcionando como prefixo e pelo nome que indica quais dados estão sendo mostrados, por exemplo: OBJECTS para os objetos, TABLES para tabelas e visões, VIEWS apenas para visões, INDEXES para índices e sucessivamente.

- Visualize as tabelas criadas pelo usuário:

```
SELECT table_name
FROM   user_tables;
```

- Visualize os tipos de objetos distintos criados pelo usuário:

```
SELECT DISTINCT object_type
FROM   user_objects;
```

- Visualize os objetos do tipo tabela (table) criados pelo usuário:

```
SELECT object_name, object_type
FROM   user_objects
WHERE  object_type = 'TABLE';
```

Você pode consultar as views do dicionário de dados para visualizar vários objetos do banco de dados criados por você e por outros usuários (dependendo do nível de acesso). Algumas visões bastante utilizadas são:

- USER_TABLES, ALL_TABLES ou DBA_TABLES;
- USER_OBJECTS, ALL_OBJECTS ou DBA_OBJECTS;
- USER_CATALOG, ALL_CATALOG ou DBA_CATALOG.

Nota: Além destas visões que possuem dados detalhados, ainda existem resumos como TAB, OBJ ou CAT.

Tipos de Dados

Tipo de Dado	Descrição
NUMBER(precisão, decimais) e NUMERIC(precisão, decimais)	Valor numérico de tamanho variável contendo um número de máximo de dígitos (total de dígitos) definido por (precisão), sendo o número de dígitos à direita do ponto decimal (decimais) definido por (decimais). A precisão máxima suportada é 38. Se nem a precisão nem os decimais são especificados, então um número com precisão e decimais de até 38 dígitos pode ser utilizado (significa que você pode utilizar um número de até 38 dígitos e podem estar à direita ou a esquerda do ponto decimal).
VARCHAR2(tamanho)	Valor caractere com tamanho variável e até o tamanho máximo definido por (tamanho). Tamanho máximo 4000 bytes.
CHAR(tamanho)	Valor caractere com tamanho fixo de tamanho definido por (tamanho) com preenchimento do final com espaços. Tamanho máximo 2000 bytes.
DATE	Data e hora (dia, mês, ano, hora, minuto e segundo) incluindo o século entre 1 de Janeiro de 4712 A.C. e 31 de dezembro de 9999 D.C. O formato default de apresentação de data é definido pelo parâmetro NLS_DATE_FORMAT (por exemplo: DD/MM/YY).
BINARY_FLOAT	Introduzido no Oracle Database 10g, armazena um numérico de precisão simples (32-bits ponto flutuante). Operações envolvendo BINARY_FLOAT são tipicamente executadas mais rapidamente que operações utilizando NUMBER. BINARY_FLOAT requer 5 bytes de espaço de armazenamento.
BINARY_DOUBLE	Introduzido no Oracle Database 10g, armazena um numérico de precisão dupla (64-bits ponto flutuante). Operações envolvendo BINARY_DOUBLE são tipicamente executadas mais rapidamente que operações utilizando NUMBER. BINARY_DOUBLE requer 9 bytes de espaço de armazenamento.
DEC e DECIMAL	Subtipo de NUMBER. Um número decimal de até 38 dígitos.
REAL	Subtipo de NUMBER. Um número ponto flutuante com até 18 dígitos de precisão.

INT, INTEGER, e SMALLINT	Subtipo de NUMBER. Um inteiro com até 38 dígitos de precisão.
NVARCHAR2(tamanho)	Valor caractere Unicode com tamanho variável e até o tamanho máximo definido por (tamanho). O número de bytes armazenados é 2 multiplicado por (tamanho) para codificação AL16UTF16 e 3 multiplicado por <i>(tamanho) para codificação</i> UTF8. Tamanho máximo 4000 bytes.
NCHAR(tamanho)	Valor caractere Unicode com tamanho fixo de tamanho definido por (tamanho) com preenchimento do final com espaços. O número de bytes armazenados é 2 multiplicado por (tamanho) para codificação AL16UTF16 e 3 multiplicado por <i>(tamanho) para codificação</i> UTF8. Tamanho máximo 2000 bytes.
LONG	Dados caractere de tamanho variável de até 2 gigabytes
CLOB	Dados caractere single-byte de até (4 gigabytes – 1) * tamanho do bloco de dados Oracle.
RAW(tamanho)	Dados binários com tamanho especificado por <i>tamanho</i> . Tamanho máximo é 2000 bytes (um tamanho máximo deve ser especificado.)
LONG RAW	Dados binários de tamanho variável de até 2 gigabytes
BLOB	Dados binários de até tamanho máximo de: (4 gigabytes – 1) * tamanho do bloco de dados Oracle
BFILE	Ponteiro para um arquivo externo de tamanho máximo de até 4Gb
ROWID	String representado em Base 64 string que especifica o endereço único de uma linha na tabela. Este tipo de dado é retornado pela pseudocoluna ROWID.
UROWID [(tamanho)]	String representado em Base 64 que especifica o endereço lógico de uma linha de uma index-organized table. O tamanho opcional é o <i>tamanho da coluna de tipo UROWID</i> . O tamanho máximo e o tamanho default é 4000 bytes.

Figura 9-2: Tipos de Dados

Criando uma Tabela Utilizando uma Sub-consulta

Um segundo método para criar uma tabela é aplicar a cláusula AS subquery para criar a tabela e já inserir as linhas retornadas pela Sub-consulta.

Sintaxe:

```
CREATE TABLE tabela
    [coluna1 (, coluna2...)]
    AS sub-consulta;
```

tabela: é o nome da tabela.

coluna: é o nome da coluna, valor default e constraints de integridade.

sub-consulta: é o comando SELECT que define o conjunto de linhas a ser inserido na tabela nova.

Diretrizes

- A tabela será criada com os nomes de coluna especificados, e as linhas recuperadas pelo comando SELECT serão inseridas na tabela.
- A definição da coluna pode conter somente o nome e o valor default.
- Se as especificações de coluna forem determinadas, o número de colunas deve ser igual ao número de colunas da lista da cláusula SELECT da Sub-consulta.
- Se nenhuma especificação de coluna é determinada, os nomes das colunas da tabela serão iguais aos nomes de coluna da Sub-consulta.

Criando uma Tabela a Partir de uma sub-consulta

```
CREATE TABLE tcontratos_vip
AS
SELECT id, tclientes_id, dt_compra, desconto, total
FROM tcontratos
WHERE total >= 5000;
```

O exemplo cria uma tabela, TCONTRATOS_VIP, que contém detalhes de todos os contratos que possuem total maior ou igual a R\$5000. Observe que os dados para a tabela TCONTRATOS_VIP estão sendo obtidos a partir da tabela TCONTRATOS.

Comando ALTER TABLE

```
ALTER TABLE  tabela
ADD (coluna1 tipo_de_dado [DEFAULT expressão]
    [, coluna2 tipo_de_dado]...);
```

```
ALTER TABLE  tabela
MODIFY (coluna1 tipo_de_dado [DEFAULT expressão]
    [, coluna2 tipo_de_dado]...);
```

```
ALTER TABLE  tabela
DROP COLUMN coluna;
```

```
ALTER TABLE  tabela
RENAME COLUMN coluna_anterior to coluna_novo;
```

Depois de criar as suas tabelas, você pode necessitar mudar a estrutura da tabela porque você omitiu uma coluna ou sua definição de coluna precisa ser modificada, ou ainda excluir uma coluna que você julga sem importância. Você pode fazer isto utilizando o comando ALTER TABLE.

Você pode adicionar colunas para uma tabela utilizando o comando ALTER TABLE com a cláusula ADD.

Sintaxe:

tabela: é o nome da tabela.

coluna: é o nome da nova coluna.

Tipo_de_dado: é o tipo de dado e tamanho da nova coluna.

DEFAULT expressão: especifica o valor default para uma nova coluna.

Você pode modificar colunas existentes em uma tabela utilizando o comando ALTER TABLE com a cláusula MODIFY.

Você pode excluir colunas existentes em uma tabela utilizando o comando ALTER TABLE com a cláusula DROP.

Adicionando uma Coluna

O exemplo:

```
ALTER TABLE tclientes  
ADD (pais VARCHAR2(30));
```

Diretrizes para Adicionar uma Coluna

- Você utiliza a cláusula ADD para adicionar colunas:
- A nova coluna torna-se a última coluna:
- Você não pode especificar onde a coluna irá aparecer. A nova coluna se torna a última coluna.

Nota: Se uma tabela já possui linhas quando uma coluna é adicionada, então a nova coluna é inicialmente nula em todas as linhas. A não ser que seja definido um valor DEFAULT para a coluna.

Modificando uma Coluna

Exemplo:

```
ALTER TABLE tdescontos  
MODIFY (classe VARCHAR2(3));
```

Você pode modificar uma definição de coluna utilizando o comando ALTER TABLE com a cláusula MODIFY. A modificação de coluna pode incluir mudanças para o tipo de dado, o tamanho e valor default.

Diretrizes

- Aumente o tamanho ou precisão de uma coluna numérica.
- Diminua o tamanho de uma coluna se ela possuir somente valores nulos ou se a tabela não possuir nenhuma linha.
- Modifique o tipo de dado se a coluna possuir somente valores nulos.
- Converta uma coluna do tipo de dado CHAR para VARCHAR2 ou converta uma coluna VARCHAR2 para o tipo de dado CHAR se a coluna possuir somente valores nulos ou se você não modificar o tamanho.
- Uma mudança para o valor default de uma coluna afeta somente as inserções subsequentes para a tabela.

Removendo uma Coluna

O comando ALTER TABLE também pode remover uma coluna da tabela
Exemplo:

```
ALTER TABLE table DROP COLUMN column;
```

Diretrizes

- A coluna e todos os dados da coluna são removidos.

Renomeando uma Coluna

Exemplo:

```
ALTER TABLE tcursos  
RENAME COLUMN carga_horaria to duracao;
```

Diretrizes

- O nome da coluna é renomeado.
- Todos os dados da coluna são mantidos.
- Definições de índices, constraints e privilégios são mantidas.

ALTER TABLE READY ONLY

O comando Table pode ser utilizado para colocar uma tabela somente para consulta.
Sintaxe:

```
ALTER TABLE tabela READ ONLY;
```

Onde:

tabela: é o nome da tabela.

Diretrizes

- A tabela somente poderá ser utilizada para operações de consulta.
- Não serão permitidas operações de INSERT, UPDATE e DELETE na tabela.

Exemplo:

```
ALTER TABLE TDESCONTOS READ ONLY;
```

ALTER TABLE READY READ WRITE

O comando Table pode seu utilizado para colocar uma tabela para permitir operações de escrita (INSERT, UPDATE, DELETE).

Sintaxe:

```
ALTER TABLE tabela READ WRITE;
```

Onde:

tabela: é o nome da tabela.

Diretrizes

- Serão permitidas operações de escrita (INSERT, UPDATE e DELETE) na tabela.
- Serão permitas operações de consulta (SELECT) na tabela.

Exemplo:

```
ALTER TABLE TDESCONTOS READ WRITE;
```

Renomeando uma Tabela

Sintaxe:

```
ALTER TABLE nometabela_anterior  
RENAME to nometabela_novo;
```

Exemplo:

```
ALTER TABLE tcursos  
RENAME to cursos;
```

Diretrizes

- O nome da tabela é renomeado.
- Todos os dados da tabela são mantidos.
- Definições de índices, constraints e privilégios são mantidas.

Renomeando um Objeto

Sintaxe:

```
RENAME nome_anterior TO nome_novo;
```

Exemplo:

```
RENAME tcursos to cursos;
```

Diretrizes

- O nome do objeto é renomeado.

Truncando uma Tabela

Sintaxe:

```
TRUNCATE TABLE tabela;
```

Onde:

tabela: é o nome da tabela.

Exemplo:

```
TRUNCATE TABLE tclientes;
```

Outro comando DDL é o comando TRUNCATE TABLE, utilizado para remover todas as linhas de uma tabela e liberar o espaço de armazenamento utilizado por ela. Quando utilizar o comando TRUNCATE TABLE, você não pode efetuar rollback das linhas removidas.

Você deve ser o dono da tabela ou possuir o privilégio de sistema DELETE TABLE para truncar uma tabela.

O comando DELETE também pode remover todas as linhas de uma tabela, porém ele não libera o espaço de armazenamento ocupado por ela.

Adicionando Comentários para Tabelas e Colunas

Sintaxe:

```
COMMENT ON TABLE tabela | COLUMN tabela.coluna IS 'texto';
```

Onde:

tabela: é o nome da tabela.

coluna: é o nome da coluna de uma tabela.

texto: é o texto do comentário.

```
COMMENT ON TABLE tclientes IS 'Informações de Clientes';
```

```
COMMENT ON COLUMN tclientes.id IS 'Código do Cliente';
```

Você pode adicionar um comentário de até 2000 bytes sobre uma coluna, tabela, visão ou snapshot utilizando o comando COMMENT. O comentário é armazenado no dicionário de dados e pode ser visualizado em uma das seguintes visões do dicionário de dados na coluna COMMENTS:

- ALL_COL_COMMENTS
- USER_COL_COMMENTS
- ALL_TAB_COMMENTS
- USER_TAB_COMMENTS

Você pode remover um comentário do banco de dados atribuindo uma string vazia ('').

Exercícios – 11

1. No SQL Developer utilize a sua conexão e conecte-se ao banco de dados do curso. Crie a tabela PESSOAS baseado na descrição abaixo.

Column Name	Data Type	Nullable
ID	NUMBER(7,0)	Yes
NOME	VARCHAR2(25 BYTE)	Yes

2. Insira linhas na tabela PESSOAS com dados a partir das linhas da tabela TCLIENTES. Inclua somente as colunas necessárias (id, nome).
3. Crie a tabela CONTRATOS baseado na descrição abaixo.

Column Name	Data Type	Nullable
ID	NUMBER(5,0)	Yes
PESSOA_ID	NUMBER(7,0)	Yes
DATA	DATE	Yes
DESCONTO	NUMBER(7,2)	Yes
TOTAL	NUMBER(7,2)	Yes

4. Modifique a coluna ID da tabela CONTRATOS para permitir o uso de valores numéricos de até 7 dígitos (id NUMBER (7)).
5. Confirme que as tabelas PESSOAS e CONTRATOS estão armazenadas no dicionário de dados (USER_TABLES).
6. Adicione um comentário para a definição das tabelas PESSOAS e CONTRATOS que as descreva. Confirme sua adição no dicionário de dados.

Espaço para anotações

12. Implementando Constraints

Objetivos

- Descrever constraints de integridade;
- Criar e administrar constraints.

O Que são Constraints?

O Servidor Oracle utiliza constraints para prevenir entrada de dados inválidos em tabelas.

Você pode utilizar constraints para fazer o seguinte:

- Garantir regras à nível de tabela sempre que uma linha for inserida, atualizada ou apagada desta tabela. A constraint deve ser satisfeita para a operação para ter sucesso.
- Prevenir a exclusão de uma tabela se existirem dependências em outras tabelas.

Constraints de Integridade de Dados

Constraint	Descrição
NOT NULL	Especifica que a coluna não pode conter valores nulos
UNIQUE KEY	Especifica uma coluna ou combinação de colunas cujos valores devem ser únicos para todas as linhas da tabela
PRIMARY KEY	Identifica de forma única cada linha da tabela
FOREIGN KEY	Estabelece e garante um relacionamento de chave estrangeira entre a coluna e uma coluna da tabela referenciada
CHECK	Especifica uma condição que deve ser verdadeira

Tabela 10-1: Constraints de integridade de dados

Diretrizes para Constraints

- Todas as constraints são armazenadas no dicionário de dados. As constraints são fáceis de referenciar se você tiver fornecido um nome significativo.
- Nomes de constraints devem seguir as regras padrão de nomenclatura de objetos.
- Se você não fornecer um nome para a constraint, o Oracle gera um nome com o formato SYS_Cn, onde n é um inteiro para criar um nome de constraint único.
- Constraints podem ser definidas no momento de criação da tabela ou adicionadas depois que a tabela for criada.
- Você pode visualizar as constraints definidas para uma tabela específica consultando a tabela do dicionário de dados USER_CONSTRAINTS.
- Constraints podem ser desabilitadas e habilitadas.

Constraints podem ser definidas constraints em dois níveis diferentes

Nível da Constraint	Descrição
Coluna	Referencia uma única coluna e é definida dentro da especificação da própria coluna; pode-se definir qualquer tipo de constraint de integridade.
Tabela	Referencia uma ou mais colunas e é definida separadamente das definições de colunas na tabela; pode-se definir qualquer tipo de constraint, exceto NOT NULL.

Tabela 10-2: Nível da constraint

Constraint NOT NULL

A constraint NOT NULL assegura que valores nulos não são permitidos na coluna. Colunas sem a constraint NOT NULL podem conter valores nulos por default.

A constraint NOT NULL pode ser especificada somente ao nível de coluna, e não em nível de tabela.

Sintaxe:

```
CREATE TABLE schema.tabela(
    coluna1 tipo_de_dado(tamanho) [NOT NULL] [DEFAULT
expressão]
    [,coluna2 tipo_de_dado(tamanho) ] [NOT NULL] [DEFAULT
expressão]
    [, coluna3 ..]
);
```

Onde:

schema: é o nome do usuário dono.

tTabela: é o nome da tabela.

DEFAULT expressão: especifica um valor default se um valor for omitido no comando INSERT.

coluna: é o nome da coluna.

Tipo_de_dado: é o tipo de dado e tamanho da coluna.

Exemplo:

```
CREATE TABLE tclientes(
    id                NUMBER(6)      NOT NULL,
    nome              VARCHAR2(35) NOT NULL,
    dt_nascimento     DATE,
    endereco          VARCHAR2(40),
    cidade            VARCHAR2(30),
    estado            VARCHAR2(2),
    cep               VARCHAR2(8),
    telefone          VARCHAR2(10),
    comentarios       VARCHAR2(1000);
```

O exemplo acima aplica a constraint NOT NULL para as colunas ID e NOME da tabela TCLIENTES. Uma vez que estas constraints não possuem nomes definidos, o Servidor Oracle criará nomes para elas.

Você pode especificar o nome de uma constraint na sua própria especificação:

Exemplo:

```
... nome VARCHAR2(35) CONSTRAINT tcl_nome_nn NOT NULL ...
```

Nota: Todos os exemplos de constraints descritos neste capítulo podem não estar presentes nas tabelas de exemplo fornecidas com este curso. Se desejar, estas constraints podem ser adicionadas as tabelas.

Constraint PRIMARY KEY

Uma constraint PRIMARY KEY cria uma chave primária para a tabela. Somente uma chave primária pode ser criada para cada tabela. A constraint PRIMARY KEY é uma coluna ou conjunto de colunas que identificam de forma única cada linha em uma tabela. Esta constraint obriga a unicidade da coluna ou combinação de colunas e assegura que nenhuma coluna que faça parte da chave primária possa conter um valor nulo.

Constraints PRIMARY KEY podem ser definidas ao nível de coluna ou ao nível de tabela. Uma chave primária composta é criada utilizando a definição ao nível de tabela.

Definição de constraint de Primary Key a nível de coluna

Sintaxe:

```
CREATE TABLE tabela(  
  col1 tipo_de_dado(tamanho) [NOT NULL] [default  
expressão]  
      constraint nome_constraint PRIMARY KEY,  
  [, col2 tipo_de_dado(tamanho) [NOT NULL] [default  
expressão] ..]);
```

Exemplo:

```
CREATE TABLE tclientes(  
  id          NUMBER(6)      NOT NULL  
      CONSTRAINT tclientes_id_pk PRIMARY KEY,  
  nome        VARCHAR2(35) NOT NULL,  
  dt_nascimento DATE,  
  endereco    VARCHAR2(40),  
  cidade      VARCHAR2(30),  
  estado      VARCHAR2(2),  
  cep         VARCHAR2(8),  
  telefone    VARCHAR2(10),  
  comentarios  VARCHAR2(1000));
```

Definição de constraint de Primary Key a nível de tabela

Sintaxe:

```
CREATE TABLE tabela(  
  col1 tipo_de_dado(tamanho) [NOT NULL] [default  
expressão]  
  [, col2 tipo_de_dado(tamanho) [NOT NULL] [default  
expressão] ..],  
  constraint nome_constraint PRIMARY KEY (col1 [col2, ..]));
```

Exemplo:

```
CREATE TABLE tclientes(  
  id          NUMBER(6)      NOT NULL,  
  nome        VARCHAR2(35)  NOT NULL,  
  dt_nascimento DATE,  
  endereco    VARCHAR2(40) ,  
  cidade      VARCHAR2(30) ,  
  estado      VARCHAR2(2) ,  
  cep         VARCHAR2(8) ,  
  telefone    VARCHAR2(10) ,  
  comentarios VARCHAR2(1000) ,  
  CONSTRAINT tclientes_id_pk PRIMARY KEY(id));
```

O exemplo acima define uma constraint PRIMARY KEY na coluna ID da tabela TCLIENTES. O nome da constraint é definido como TCLIENTES_ID_PK.

Nota: Um índice único é criado automaticamente para uma coluna de chave primária.

Constraint UNIQUE KEY

Uma constraint de integridade do tipo UNIQUE faz com que cada valor em uma coluna ou conjunto de colunas (chave) seja único, ou seja, duas linhas de uma tabela não podem ter valores duplicados na coluna especificada ou no conjunto de colunas. A coluna (ou conjunto de colunas) incluída na definição da constraint UNIQUE é chamada de chave única. Se uma chave única possuir mais de uma coluna, o grupo de colunas será chamado de chave única composta.

Constraints UNIQUE permitem a inserção de nulos a menos que você também defina uma constraint NOT NULL para as mesmas colunas. De fato, qualquer número de linhas pode receber nulos para colunas sem a constraint NOT NULL porque nulos não são considerados iguais a nada. Um nulo em uma coluna (ou em todas as colunas de uma chave única composta) sempre satisfaz a constraint UNIQUE.

Nota: Devido ao mecanismo de procura das constraints UNIQUE com mais de uma coluna, você não pode ter valores idênticos nas colunas não nulas de uma constraint de chave única composta parcialmente nula.

Constraints UNIQUE podem ser definidas ao nível de coluna ou tabela. Uma chave única composta é criada utilizando a definição ao nível de tabela.

Definição de constraint de UNIQUE Key a nível de coluna

Sintaxe:

```
CREATE TABLE tabela(  
  col1 tipo_de_dado(tamanho) [NOT NULL] [default expressão]  
      constraint nome_constraint UNIQUE,  
  [, col2 tipo_de_dado(tamanho) [NOT NULL] [default expressão] ..]);
```

Exemplo: Definição de constraint Unique Key em nível de coluna:

```
CREATE TABLE tcursos(  
  id                NUMBER(4) NOT NULL CONSTRAINT tcs_id_pk PRIMARY KEY,  
  nome              VARCHAR2(100),  
  cod_trg           VARCHAR2(8) CONSTRAINT tcs_cod_trg_uk UNIQUE,  
  preco             NUMBER(7,2),  
  carga_horaria     NUMBER(3),  
  dt_criacao        DATE,  
  pre_requisito     NUMBER(4));
```

Definição de constraint de Unique Key a nível de tabela

Sintaxe:

```
CREATE TABLE tabela(  
  col1 tipo_de_dado(tamanho) [NOT NULL] [default  
expressão]  
  [, col2 tipo_de_dado(tamanho) [NOT NULL] [default  
expressão] ..],  
  constraint nome_constraint UNIQUE KEY (col1 [col2, ..]));
```

Exemplo: Definição de constraint Unique Key a nível de tabela:

```
CREATE TABLE tcursos(  
  id                NUMBER(4) NOT NULL,  
  nome              VARCHAR2(100),  
  cod_trg           VARCHAR2(8),  
  preco             NUMBER(7,2),  
  carga_horaria     NUMBER(3),  
  dt_criacao        DATE,  
  pre_requisito     NUMBER(4),  
  CONSTRAINT tcs_id_pk PRIMARY KEY (id),  
  CONSTRAINT tcs_cod_trg_uk UNIQUE (cod_trg));
```

Nota: O Servidor Oracle implementa a constraint UNIQUE criando implicitamente um índice único na chave única.

Constraint FOREIGN KEY

Uma FOREIGN KEY, ou constraint de integridade referencial, designa uma coluna ou combinação de colunas como chave estrangeira e estabelece uma relação entre uma chave primária ou uma chave única para a mesma tabela ou para uma tabela diferente.

Um valor de chave estrangeira deve corresponder a um valor existente na tabela pai ou deve ser nulo.

Chaves estrangeiras são baseadas nos valores dos dados e são puramente lógicas, não físicas ou ponteiros.

Constraints FOREIGN KEY podem ser definidas ao nível de coluna ou de tabela. Uma chave estrangeira composta é criada utilizando a definição ao nível de tabela.

Definição de constraint de FOREIGN Key a nível de coluna

Sintaxe:

```
CREATE TABLE tabela(  
  col1 tipo_de_dado(tamanho) [NOT NULL] [default  
expressão]  
  constraint nome_constraint FOREIGN KEY (col1,  
col2..])  
  REFERENCES nome_tabela (col1 [, col2..]),  
  [, col2 tipo_de_dado(tamanho) [NOT NULL] [default  
expressão] ..]);
```

Exemplo: Definição de constraint FOREIGN Key a nível de coluna:

```
CREATE TABLE tcontratos(  
  id          NUMBER(4) NOT NULL  
  CONSTRAINT tcontratos_id_pk PRIMARY KEY,  
  dt_compra   DATE,  
  status      VARCHAR2(1),  
  tclientes_id NUMBER(6) NOT NULL  
  CONSTRAINT tcn_tclientes_id_fk  
  REFERENCES tclientes(id),  
  desconto   NUMBER(7,2),  
  total       NUMBER(10,2));
```

Definição de constraint de Foreign Key a nível de tabela

Sintaxe:


```
CREATE TABLE tabela(  
  col1 tipo_de_dado(tamanho) [NOT NULL] [default  
expressão]  
  [, col2 tipo_de_dado(tamanho) [NOT NULL] [default  
expressão] ..],  
  constraint constraint nome_constraint FOREIGN KEY (col1 [,  
col2..]) REFERENCES nome_tabela (col1 [, col2..]));
```

Exemplo: Definição de constraint Foreign Key a nível de tabela:

```
CREATE TABLE tcontratos(  
  id          NUMBER(4) NOT NULL,  
  dt_compra   DATE,  
  status      VARCHAR2(1),  
  tclientes_id NUMBER(6) NOT NULL,  
  desconto    NUMBER(7,2),  
  total       NUMBER(10,2),  
  CONSTRAINT tcn_tclientes_id_fk FOREIGN KEY(tclientes_id)  
            REFERENCES tclientes(id),  
  CONSTRAINT tcontratos_id_pk PRIMARY KEY(id));
```

O exemplo acima define uma constraint de chave estrangeira na coluna TCLIENTES_ID da tabela TCONTRATOS. O nome da constraint é definido como TCN_TCLIENTES_ID_FK.

Palavras Chave de Constraints FOREIGN KEY

A chave estrangeira é definida na tabela dependente (filha), e a tabela que contém a coluna referenciada é a tabela pai. Uma chave estrangeira é definida utilizando-se uma combinação das seguintes palavras chaves:

- **FOREIGN KEY:** utilizada para definir a coluna da tabela filha quando a constraint for definida ao nível de tabela.
- **REFERENCES:** identifica a tabela e coluna da tabela referenciada (pai).
- **ON DELETE CASCADE:** indica que quando uma linha da tabela pai é removida, as linhas dependentes da tabela filha também serão apagadas.
- **ON DELETE SET NULL:** indica que quando uma linha da tabela pai é removida, os valores da chave estrangeira (foreign key) serão alterados para nulo (NULL).

Sem a opção ON DELETE CASCADE ou ON DELETE SET NULL uma linha da tabela pai não pode ser removida se estiver sendo referenciada na tabela filha.

Constraint CHECK

A constraint CHECK define uma condição que cada linha deve satisfazer. A condição pode utilizar a mesma construção que as condições de consultas, com as seguintes exceções:

- Referências para as pseudo colunas CURRVAL, NEXTVAL, LEVEL e ROWNUM.
- Chamadas para as funções SYSDATE, UID e USER.
- Consultas que referenciam outros valores em outras linhas.

Uma única coluna pode ter múltiplas constraints tipo CHECK que referenciam a definição da coluna. Não há nenhum limite quanto ao número de constraints tipo CHECK que você pode definir em uma coluna.

Constraints CHECK podem ser definidas ao nível de coluna ou a nível de tabela.

Definindo Check Constraints a nível de coluna.

Exemplo:

```
CREATE TABLE tcontratos(
  id          NUMBER(4) NOT NULL,
  dt_compra   DATE,
  status      VARCHAR2(1),
  tclientes_id NUMBER(6) NOT NULL,
  desconto    NUMBER(7,2)
    CONSTRAINT tcn_desconto_ck CHECK(desconto BETWEEN 0 AND 1000),
  total       NUMBER(10,2),
  CONSTRAINT tcn_tclientes_id_fk FOREIGN KEY(tclientes_id)
    REFERENCES tclientes(id),
  CONSTRAINT tcontratos_id_pk PRIMARY KEY(id));
```

Definindo Check Constraints a nível de tabela

Exemplo:

```
CREATE TABLE tcontratos(
  id          NUMBER(4) NOT NULL,
  dt_compra   DATE,
  status      VARCHAR2(1),
  tclientes_id NUMBER(6) NOT NULL,
  desconto    NUMBER(7,2)
    CONSTRAINT tcn_desconto_ck CHECK(desconto BETWEEN 0 AND 1000),
  total       NUMBER(10,2),
  CONSTRAINT tcn_tclientes_id_fk FOREIGN KEY(tclientes_id)
    REFERENCES tclientes(id),
  CONSTRAINT tcontratos_id_pk PRIMARY KEY(id),
  CONSTRAINT tcn_desconto_ck CHECK(desconto BETWEEN 0 AND 1000));
```

Adicionando uma Constraint

```
ALTER TABLE tabela ADD [CONSTRAINT constraint]  
Tipo_constraint (col1, [, col2..]);
```

Você pode adicionar uma constraint para tabelas existentes utilizando o comando ALTER TABLE com a cláusula ADD.

Sintaxe:

tabela: é o nome da tabela.

constraint: é o nome da constraint.

Tipo_constraint: é o tipo de constraint.

col1, *col2*: é o nome das colunas afetada sobre as quais a constraint foi definida.

A sintaxe do nome da constraint é opcional, embora recomendada. Se você não fornecer um nome para suas constraints, o sistema gerará nomes para elas.

Diretrizes

- Você pode adicionar, remover, habilitar ou desabilitar uma constraint, mas você não pode modificar sua estrutura.
- Você pode adicionar uma constraint NOT NULL para uma coluna existente utilizando a cláusula MODIFY do comando ALTER TABLE.

Nota: Você pode definir uma coluna NOT NULL somente se a tabela não possuir nenhuma linha porque não podem ser especificados dados para linhas existentes ao mesmo tempo em que a coluna é adicionada.

```
ALTER TABLE tcursos  
ADD CONSTRAINT tcursos_id_pre_requisito_fk  
FOREIGN KEY(pre_requisito) REFERENCES tcursos(id);
```

O exemplo acima cria uma constraint FOREIGN KEY na tabela TCURSOS. A constraint garante que um curso possua um pré-requisito válido na tabela TCURSOS.

Removendo uma Constraint

Sintaxe:

```
ALTER TABLE tabela
DROP PRIMARY KEY | UNIQUE (coluna [, coluna..])
| CONSTRAINT constraint
[CASCADE];
```

Onde:

tabela: é o nome da tabela.

coluna: é o nome da coluna afetada pela constraint.

constraint: é o nome da constraint.

Exemplo:

- Remova a constraint de pré-requisitos da tabela TCURSOS:

```
ALTER TABLE tcursos DROP CONSTRAINT tcursos_id_pre_requisito_fk;
```

- Remova a constraint PRIMARY KEY da tabela TCLIENTES e remova em cascata a(s) constraint(s) FOREIGN KEY que referenciam uma constraint de PRIMARY KEY.

```
ALTER TABLE tclientes DROP PRIMARY KEY CASCADE;
```

Para remover uma constraint, você pode identificar o seu nome a partir das visões do dicionário de dados USER_CONSTRAINTS e USER_CONS_COLUMNS. Então utilize o comando ALTER TABLE com a cláusula DROP. A opção CASCADE da cláusula DROP também remove qualquer constraint dependente.

Quando você remove uma constraint de integridade, esta constraint não mais é verificada pelo Servidor Oracle e não fica mais disponível no dicionário de dados.

Desabilitando Constraints

Sintaxe:

```
ALTER TABLE tabela  
DISABLE CONSTRAINT constraint [CASCADE];
```

Onde:

tabela: é o nome da tabela.

constraint : é o nome da constraint.

```
ALTER TABLE tdescontos  
DISABLE CONSTRAINT tds_classe_pk CASCADE;
```

Você pode desabilitar uma constraint sem removê-la ou recriá-la utilizando o comando ALTER TABLE com a cláusula DISABLE.

Diretrizes

- Você pode utilizar a cláusula DISABLE no comando CREATE TABLE e no comando ALTER TABLE.
- A cláusula CASCADE desabilita as constraints de integridade dependentes em cascata.

Habilitando Constraints

Sintaxe:

```
ALTER TABLE tabela ENABLE CONSTRAINT constraint;
```

Onde:

tabela: é o nome da tabela.

constraint: é o nome da constraint.

```
ALTER TABLE tclientes ENABLE CONSTRAINT tclientes_id_pk;
```

Você pode habilitar uma constraint sem removê-la ou recriá-la utilizando o comando ALTER TABLE com a cláusula ENABLE.

Diretrizes

- Se você habilita uma constraint, esta constraint aplica-se a todos os dados da tabela. Todos os dados da tabela devem ajustar-se a constraint.
- Se você habilita uma constraint UNIQUE key ou PRIMARY KEY, um índice do tipo UNIQUE ou PRIMARY KEY é criado automaticamente.
- Você pode utilizar a cláusula ENABLE no comando CREATE TABLE e no comando ALTER TABLE.

Visualizando Constraints

Após criar uma tabela, você pode confirmar sua existência executando um comando DESCRIBE. A única constraint que você pode verificar desta forma é a constraint NOT NULL. Para visualizar todas as constraints de sua tabela, consulte a tabela USER_CONSTRAINTS.

Nota: Constraints que não receberam nomes na sua criação recebem um nome atribuído pelo sistema. Na coluna CONSTRAINT_TYPE:

- "C" representa CHECK
- "P" representa PRIMARY KEY
- "R" representa integridade referencial (FOREIGN KEY)
- "U" representa UNIQUE key.

Observe que as constraints NOT NULL na verdade são constraints CHECK.

Exemplo:

```
SELECT  constraint_name, constraint_type, search_condition
FROM    user_constraints
WHERE   table_name = 'TCLIENTES';
```

O exemplo acima exibe todas as constraints da tabela TCLIENTES.

Visualizando as Colunas Associadas com Constraints

Você pode visualizar os nomes de colunas envolvidas em constraints consultando a visão do dicionário de dados USER_CONS_COLUMNS. Esta visão é especialmente útil para constraints que possuem o nome atribuído pelo sistema.

Exemplo:

```
SELECT constraint_name, column_name
FROM   user_cons_columns
WHERE  table_name = 'TCONTRATOS';
```

Exercícios – 12

1. No SQL Developer utilize a sua conexão e conecte-se ao banco de dados do curso. Adicione uma constraint PRIMARY KEY para a tabela CONTRATOS utilizando a coluna ID.
2. Crie uma constraint PRIMARY KEY na tabela PESSOAS utilizando a coluna ID.
3. Adicione uma referência de chave estrangeira para a tabela CONTRATOS que garanta que o contrato não seja associado a uma pessoa com ID inexistente (coluna PESSOAS_ID).
4. Confirme que as constraints foram adicionadas consultando a visão USER_CONSTRAINTS. Observe os tipos e nomes das constraints.
5. Modifique a tabela CONTRATOS. Adicione uma coluna IMPOSTO com o tipo de dado NUMBER(7,2) definindo que o valor do imposto não pode ser negativo.

Espaço para anotações

13. Criando Visões

Objetivos

- Descrever uma visão;
- Criar uma visão;
- Recuperar dados através de uma visão;
- Alterar a definição de uma visão;
- Inserir, atualizar e remover dados através de uma visão;
- Remover uma visão.

O que é uma Visão?

Você pode apresentar subconjuntos lógicos ou combinações de dados criando visões das tabelas. Uma visão é uma representação lógica baseada em um SELECT sobre uma tabela ou mais tabelas ou outras visões. Uma visão não possui dados próprios, mas é como uma janela pela qual os dados das tabelas podem ser visualizados ou modificados.

As tabelas nas quais uma visão é baseada são chamadas de tabelas básicas. A visão é armazenada como um comando SELECT no dicionário de dados.

Porquê Utilizar Visões?

- Para restringir o acesso ao banco de dados.
- Para tornar simples consultas complexas.
- Para permitir independência de dados.
- Para apresentar visões diferentes do mesmo dado.

Vantagens de Visões

- Restringem o acesso para o banco de dados porque a visão pode exibir uma porção seletiva do banco de dados.
- Permitem aos usuários fazer consultas simples para recuperar os resultados de consultas complexas. Por exemplo, visões permitem aos usuários consultar informações de múltiplas tabelas sem saber escrever um comando de join.
- Provê independência dos dados para os usuários e programas de aplicação. Uma visão pode ser utilizada para recuperar dados de várias tabelas.
- Provê acesso aos dados para grupos de usuários de acordo com seus critérios particulares.

Visões Simples e Visões Complexas

Característica	Visão Simples	Visão Complexa
Número de tabelas	Uma	Uma ou mais
Contém funções	Não	Sim
Possui grupos de dados	Não	Sim
DML através da visão	Sim	Talvez

Figura 11-2: Tipos de visões

Existem duas classificações para as visões: simples e complexas. A diferença básica está relacionada as operações DML (INSERT, UPDATE e DELETE).

Uma visão simples:

- Deriva dados de uma única tabela
- Não utiliza funções de grupo ou grupos de dados
- Não utiliza DISTINCT
- Pode executar operações DML através da visão

Uma visão complexa:

- Deriva dados de várias tabelas ou
- Utiliza funções ou grupos de dados ou
- Talvez permita operações DML através da visão (quando a visão completa não usa funções de grupo, group by ou distinct).

Criando uma Visão

Você pode criar uma visão inserindo uma Sub-consulta dentro do comando Sintaxe:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW nome_view [(alias[,  
alias]...)]  
AS sub-consulta  
[WITH CHECK OPTION [CONSTRAINT constraint]]  
[WITH READ ONLY]
```

Onde:

CREATE OR REPLACE: recria a visão caso ela já exista.

FORCE: cria a visão mesmo que as tabelas básicas não existam.

NOFORCE: cria a visão somente se as tabelas básicas existem. Este é o default.

nome_view: é o nome da visão.

alias: especifica nomes para as expressões selecionadas pela consulta da visão. O número de alias deve corresponder ao número de expressões selecionadas pela visão.

sub-consulta: é um comando SELECT completo. Você pode utilizar alias para as colunas na lista da cláusula SELECT.

WITH CHECK OPTION: especifica que somente as linhas acessíveis para a visão podem ser inseridas ou atualizadas.

constraint é o nome atribuído a constraint da cláusula CHECK OPTION.

WITH READ ONLY: assegura que nenhuma operação DML possa ser executada nesta visão.

Exemplo:

```
CREATE OR REPLACE VIEW vcontratos_top  
AS  
SELECT id, desconto, total  
FROM   tcontratos  
WHERE  total > 5000;
```

O exemplo acima cria uma visão que contém os contratos com total maior que R\$5000.

Você pode exibir a estrutura de uma visão utilizando o comando DESCRIBE.

```
DESCRIBE vcontratos_top
```

Diretrizes para Criação de Visões

- A Sub-consulta que define uma visão pode conter uma sintaxe complexa do comando SELECT, incluindo joins, grupos e Sub-consultas.
- Se você não especificar um nome de constraint para a visão criada com CHECK OPTION, o sistema atribui um nome default no formato SYS_Cn.
- Você pode utilizar a opção OR REPLACE para modificar a definição da visão sem ter que removê-la e recriá-la ou ter que repassar os privilégios de objeto previamente concedidos.

Você pode controlar os nomes de colunas incluindo alias de coluna dentro da Sub-consulta.

O exemplo acima cria uma visão contendo o número do curso com o alias ID , o código Target com o alias TRG e o preço do curso como VALOR para todos os curso sem pré-requisitos.

Alternativamente, você pode controlar os nomes das colunas incluindo alias de coluna na cláusula CREATE VIEW.

Exemplo:

```
CREATE OR REPLACE VIEW vcursos_basicos
AS
SELECT id ID, cod_trg TRG, preco VALOR
FROM   tcursos
WHERE  pre_requisito IS NULL;

View criada.
```

Efetuando consultas utilizando uma Visão

```
SELECT *  
FROM   v cursos_basicos;
```

Você pode recuperar dados de uma visão da mesma forma que qualquer tabela. Você pode exibir o conteúdo de toda a visão ou apenas linhas e colunas específicas.

Consultando as Visões existentes

Visões no Dicionário de Dados

Uma vez criada a visão, você pode consultar a tabela do dicionário de dados chamada `USER_VIEWS` para obter o nome da visão e sua definição. O texto do comando `SELECT` que constitui a visão é armazenado em uma coluna tipo `LONG`.

Acesso aos Dados em Visões

Quando você acessa os dados, utilizando uma visão, o Servidor Oracle executa as seguintes operações:

- 1) Recupera a definição da visão da tabela do dicionário de dados `USER_VIEWS`.
- 2) Confere os privilégios de acesso para a tabela básica da visão.
- 3) Converte a consulta para a visão em uma operação equivalente para a tabela ou tabelas básicas. Em outras palavras, os dados são recuperados a partir das, ou uma atualização é feita para, as tabelas básicas.

Modificando uma Visão

```
CREATE OR REPLACE VIEW vcursos_basicos(id, trg, valor)
AS
SELECT id, cod_trg, preco
FROM   tcursos
WHERE  preco < 1000;
```

A opção OR REPLACE permite recriar uma visão mesmo se outra já existir com este nome, substituindo a versão anterior. Isto significa que a visão pode ser alterada sem ser removida, recriando-a e mantendo os privilégios de objeto.

Nota: Quando atribuir alias de colunas na cláusula CREATE VIEW, lembre-se que os alias são listados na mesma ordem das colunas na Sub-consulta.

Criando uma Visão Complexa

```
CREATE VIEW vcontratos (data, min, max, avg)
AS
SELECT dt_compra, MIN(total), MAX(total), AVG(total)
FROM tcontratos
GROUP BY dt_compra;
```

O exemplo acima cria uma visão complexa com os valores do menor, maior contrato e a média de todos. Observe os nomes alternativos que foram especificados para a visão. Esta é uma exigência se qualquer coluna da visão é derivada de uma função ou uma expressão.

Você pode ver a estrutura da visão utilizando o comando DESCRIBE do SQL*Plus. Mostre o conteúdo da visão executando um comando SELECT.

```
SELECT *
FROM vcontratos;
```

Removendo uma Visão

Sintaxe:

```
DROP VIEW nome_view;
```

Onde:

Nome_view: é o nome da visão.

Exemplo:

```
DROP VIEW vcursos_basicos;
```

Você utiliza o comando DROP VIEW para remover uma visão. O comando remove a definição da visão do banco de dados. Remover visões não possui nenhum efeito nas tabelas nas quais a visão estava baseada. Visões ou outras aplicações baseadas em visões apagadas tornam-se inválidas. Somente o dono ou um usuário com o privilégio DROP ANY VIEW pode remover uma visão.

Regras para Executar Operações DML em uma Visão

Você pode executar operações DML nos dados através de uma visão desde que estas operações sigam certas regras.

Removendo linhas através de Uma Visão

Você não pode remover uma linha através de uma visão que contenha um dos seguintes:

- Funções de grupo.
- Uma cláusula GROUP BY.
- A palavra chave DISTINCT.
- A pseudocoluna ROWNUM.

Atualizando linhas através de Uma Visão

Você não pode atualizar linhas através de uma visão que contenha os seguintes:

- Funções de grupo.
- Uma cláusula GROUP BY.
- A palavra chave DISTINCT.
- Colunas definidas por expressões, por exemplo: PRECO * 1.2.
- A pseudocoluna ROWNUM.

Inserindo linhas através de Uma Visão

Você não pode inserir linhas através de uma visão que contenha os seguintes:

- Funções de grupo.
- Uma cláusula GROUP BY.
- A palavra chave DISTINCT.
- Colunas definidas por expressões, por exemplo: PRECO * 1.2.
- A pseudocoluna ROWNUM.
- Se existirem colunas NOT NULL, sem um valor default, na tabela básica que não foram selecionadas pela visão.

Utilizando a Cláusula WITH CHECK OPTION

É possível executar consistências de integridade referencial através de visões. Você pode também garantir constraint ao nível de banco de dados. A visão pode ser utilizada para proteger a integridade dos dados, mas seu uso é muito limitado.

A cláusula WITH CHECK OPTION especifica que INSERTS e UPDATES executados pela visão não podem criar linhas que a visão não possa selecionar, e portanto ela permite constraints de integridade e consistências de validação de dados garantida nos dados inseridos ou atualizados.

Exemplo:

```
CREATE OR REPLACE VIEW clientes_rs
AS
SELECT *
FROM   tclientes
WHERE  estado = 'RS'
WITH CHECK OPTION CONSTRAINT vclientes_rs_ck;
```

Exemplo de Violação da Check Option:

```
UPDATE clientes_rs
SET    estado = 'SP'
WHERE  id = 100;
```

Impedindo Operações DML em Visões

Você pode assegurar que nenhuma operação DML execute sobre a visão criando ela com a opção `WITH READ ONLY`.

Exemplo:

```
CREATE OR REPLACE VIEW clientes_rs
AS
SELECT *
FROM   tclientes
WHERE  estado = 'RS'
WITH READ ONLY;
```

O exemplo acima modifica a visão `clientes_rs` para prevenir qualquer operação DML através da visão.

Qualquer tentativa de executar um comando DML na visão `clientes_rs` resultará em um erro.

Exercícios – 13

1. No SQL Developer utilize a sua conexão e conecte-se ao banco de dados do curso. Crie uma visão chamada VCONTRATOS baseada no número do contrato, data de compra e no número do cliente a partir da tabela TCONTRATOS. Modifique o cabeçalho para o código do cliente para "CLIENTE".
2. Mostre o conteúdo da visão VCONTRATOS.
3. Selecione a coluna VIEW_NAME e TEXT a partir da visão USER_VIEWS do dicionário de dados.
4. Utilizando a visão VCONTRATOS, execute uma consulta para exibir todos os clientes e datas de compras.
5. Crie uma visão chamada VCLIENTESRS que contenha o número do cliente, o nome, a cidade e o estado para todos os clientes com estado igual a 'RS'. Não permita que operações DML sejam executadas através da visão.
6. Mostre a estrutura e o conteúdo da visão VCLIENTESRS.

Espaço para anotações

14. Criando Sequências, Índices e Sinônimos

Objetivos

- Criar, alterar e utilizar sequences;
- Criar e alterar índices;
- Criar sinônimos.

O que é uma Sequence?

- Um gerador de sequências pode ser utilizado para gerar números sequenciais automaticamente para linhas em tabelas. Uma sequence é um objeto do banco de dados criado por um usuário podendo ser compartilhado por múltiplos usuários. Uma sequence é um objeto do banco de dados criado por um usuário podendo ser compartilhado por múltiplos usuários.
- Um uso típico para sequence é criar um valor para uma chave primária, que deve ser único para cada linha.
- A sequence é gerada e é incrementada por uma rotina interna do Oracle. Este objeto pode reduzir o tempo de desenvolvimento uma vez que reduz a quantidade de código de aplicação necessária para gerar uma rotina que implementa uma sequência.
- Os números das sequências são armazenados e gerados independentemente das tabelas. Portanto, a mesma sequence pode ser utilizada em tabelas diferentes.

Comando CREATE SEQUENCE

```
CREATE SEQUENCE sequence
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}]
[{ORDER | NOORDER}];
```

Automaticamente gere números sequenciais utilizando o comando CREATE SEQUENCE.

Sintaxe:

sequence: é o nome do gerador da sequência.

INCREMENT BY n: especifica o intervalo entre os números da sequência onde *n* é um inteiro. Se esta cláusula for omitida, a sequência será incrementada por 1.

START WITH n: especifica o primeiro número da sequência a ser gerado. Se esta cláusula for omitida, a sequência começará em 1.

MAXVALUE n: especifica o valor de máximo que sequência pode gerar.

NOMAXVALUE: especifica um valor de máximo de 10^{27} para uma sequence ascendente e -1 para uma sequence descendente. Esta é a opção default.

MINVALUE n: especifica o valor mínimo da sequência.

NOMINVALUE: especifica um valor mínimo de 1 para uma sequence ascendente e (10^{26}) para uma sequence descendente. Esta é a opção default.

CYCLE | NOCYCLE: especifica que a sequence continua gerando valores depois de atingir seu valor máximo ou mínimo ou que ela não deve gerar valores adicionais. NOCYCLE é a opção default.

CACHE n | NOCACHE: especifica quantos valores o Servidor Oracle deve alocar e manter em memória. Por default, o Servidor Oracle mantém 20 valores em memória (cache).

ORDER | NOORDER Default NOORDER: não garante que os valores serão gerados na ordem da requisição. ORDER garante que os valores são gerados na ordem da requisição (utilizado em aplicações RAC (Real Application Clusters)).

Criando uma Sequence

```
CREATE SEQUENCE tclientes_seq  
INCREMENT BY 1  
START WITH 201  
NOMAXVALUE  
NOCACHE  
NOCYCLE;  
  
Sequência criada.
```

O exemplo acima cria uma sequence chamada TCLIENTES_SEQ para ser utilizada para a coluna ID da tabela TCLIENTES. A sequência começa em 201, não permite cache e não permite que ela seja cíclica.

Não utilize a opção CYCLE se a sequence for utilizada para gerar valores de chave primária a menos que você possua um mecanismo que remova linhas antigas mais rapidamente que os ciclos da sequence.

Consultando as Sequences definidas

Uma vez criada, a sequence é documentada no dicionário de dados. Considerando-se que uma sequence é um objeto do banco de dados, você pode identificá-la na tabela do dicionário de dados USER_OBJECTS.

Você pode também confirmar as configurações da sequence selecionando a partir da tabela do dicionário de dados USER_SEQUENCES.

Exemplo:

```
SELECT  sequence_name, min_value, max_value, increment_by,  
        last_number  
FROM    user_sequences;
```

Pseudocolunas NEXTVAL e CURRVAL

Uma vez criada a sequence, você pode utilizá-la para gerar números sequenciais para suas tabelas. Referencie os valores da sequence utilizando as pseudocolunas NEXTVAL e CURRVAL.

A pseudocoluna NEXTVAL é utilizada para extrair sucessivos números da sequence especificada. Você deve qualificar NEXTVAL com o nome da sequence. Quando você referencia sequence.NEXTVAL, um novo número da sequence é gerado e colocado em CURRVAL.

A pseudocoluna CURRVAL é utilizada para se referenciar a um número da sequence que o usuário atual gerou anteriormente.

Antes de poder referenciar CURRVAL na sessão você deve referenciar pelo menos uma vez na sua sessão.

Você deve qualificar CURRVAL com o nome da sequence. Quando sequence.CURRVAL é referenciada, o último valor retornado ao processo daquele usuário é exibido.

Regras para Utilizar NEXTVAL e CURRVAL

Você pode utilizar NEXTVAL e CURRVAL no seguinte:

- Na lista da cláusula SELECT de um comando SELECT que não seja parte de uma Sub-consulta.
- Na lista da cláusula SELECT de uma Sub-consulta em um comando INSERT.
- Na cláusula VALUES de um comando INSERT.
- Na cláusula SET de um comando UPDATE.
- Você não pode utilizar NEXTVAL e CURRVAL no seguinte:
- Na lista da cláusula SELECT de uma visão.
- Em um comando SELECT com a palavra chave DISTINCT.
- Em um comando SELECT com as cláusulas GROUP BY, HAVING ou ORDER BY
- Em uma Sub-consulta no comando SELECT, DELETE ou UPDATE.
- Na expressão da opção DEFAULT em um comando CREATE TABLE ou ALTER TABLE.

Utilizando uma Sequence

Exemplo:

```
INSERT INTO tclientes( id, nome, dt_nascimento)
VALUES (tclientes_seq.NEXTVAL, 'Rafael Monteiro',
TO_DATE('01/02/1970', 'DD/MM/YYYY');
```

O exemplo acima insere um novo cliente na tabela TCLIENTES, utilizando a sequence SCLIENTES_ID para gerar um novo número de cliente.

Você pode visualizar o valor atual da sequence:

```
SELECT    tclientes_seq.CURRVAL
FROM      dual;
```

Mantendo em Memória os Valores da Sequence

Mantenha sequências em memória para permitir acesso mais rápido aos valores da sequence. O cache é populado na primeira referência a sequence. Cada solicitação para o próximo valor da sequence é recuperada da memória. Após a última sequência ser utilizada, a próxima solicitação para a sequence busca outro cache de sequências para a memória.

Previna Falhas na Sequencia

Embora os geradores de sequência forneçam números sequenciais sem falhas, esta ação ocorre independente de um commit ou rollback. Portanto, se você efetuar o rollback de um comando que contém uma sequence, o número será perdido.

Outro evento que pode causar falhas na numeração da sequence é uma falha do sistema. Se a sequence possui valores no cache de memória, esses valores serão perdidos na falha do sistema.

Uma vez que as sequences não são relacionadas diretamente às tabelas, a mesma sequence pode ser utilizada em múltiplas tabelas. Se isto ocorrer, cada tabela pode conter falhas nos números sequenciais.

Visualizando o Próximo Valor Disponível da Sequence sem Incrementá-lo

É possível visualizar o próximo valor disponível da sequence sem incrementá-lo, porém somente se a sequence foi criada com NOCACHE, examinando a visão do dicionário de dados USER_SEQUENCES.

Modificando uma Sequence

Sintaxe:

```
ALTER SEQUENCE sequence
[INCREMENT BY n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
[ORDER | NOORDER]
```

Se você atingir o limite MAXVALUE para uma sequence, não serão alocados valores adicionais da sequence e você receberá um erro que indica que o valor de MAXVALUE foi excedido. Para continuar utilizando a sequence, você pode modificá-la utilizando o comando ALTER SEQUENCE.

Sintaxe:

Onde:

sequence: é o nome do gerador da sequence.

Exemplo:

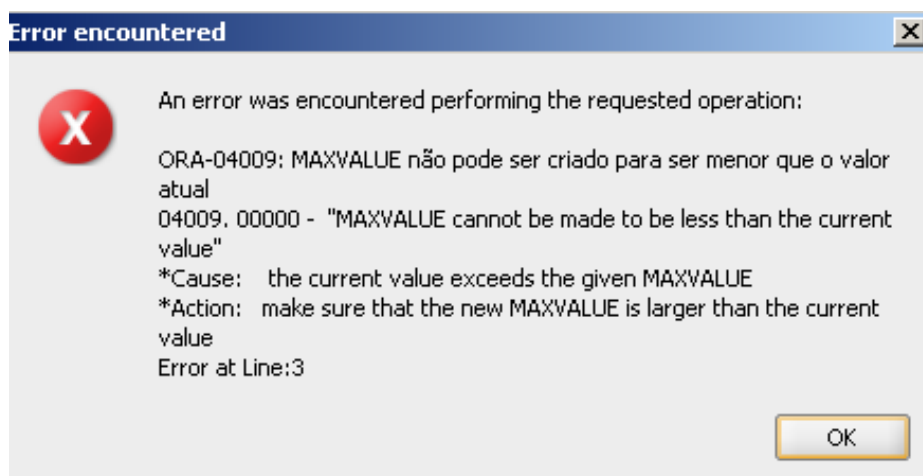
```
ALTER SEQUENCE tclientes_seq
MAXVALUE 999999;
```

Diretrizes para Modificar uma Sequence

- Você deve ser o dono ou possuir o privilégio ALTER para a sequence para poder modificá-la.
- Somente os próximos números da sequence serão afetados pelo comando ALTER SEQUENCE.
- A opção START WITH não pode ser modificada utilizando o comando ALTER SEQUENCE. A sequence deve ser removida e recriada para reiniciar em um número diferente.
- Algumas validações são executadas. Por exemplo, um novo MAXVALUE não pode ser menor que o número atual da sequence.

Exemplo:

```
ALTER SEQUENCE tclientes_seq  
MAXVALUE 90;
```



Removendo uma Sequence

Para remover uma sequence do dicionário de dados, utilize o comando DROP SEQUENCE. Você deve ser o dono da sequence ou ter o privilégio DROP ANY SEQUENCE para removê-la.

Sintaxe:

```
DROP SEQUENCE sequence;
```

Onde:

sequence: é o nome do gerador da sequência.

Exemplo:

```
DROP SEQUENCE tclientes_seq;
```

O que é um Índice?

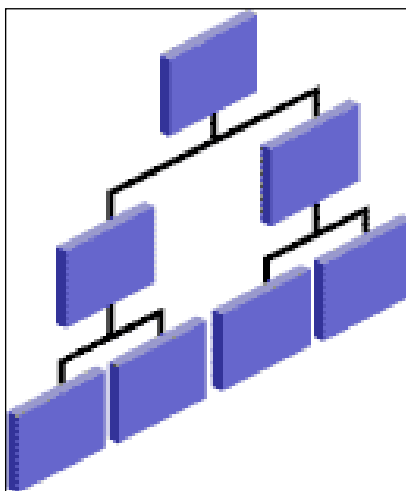
Um índice do Servidor Oracle é um objeto de um schema que pode acelerar a recuperação das linhas utilizando um ponteiro. Podem ser criados explicitamente ou automaticamente. Se você não possuir um índice em uma coluna, então um método de acesso chamado full table scan ocorrerá.

Um índice fornece acesso direto e rápido para as linhas de uma tabela. Seu propósito é reduzir a necessidade de I/O de disco utilizando um caminho indexado para localizar os dados rapidamente. O índice é automaticamente utilizado e mantido pelo Servidor Oracle. Uma vez criado, nenhuma atividade direta é requerida por parte do usuário.

Índices são lógica e fisicamente independentes da tabela que indexam. Isto significa que eles podem ser criados ou removidos a qualquer momento e não causam nenhum efeito nas tabelas ou outros índices.

Nota: Quando você remove uma tabela, os índices correspondentes também são removidos.

Estrutura de um índice BTree:



Como os Índices são Criados?

Podem ser criados dois tipos de índices. Um tipo são os índices únicos. O Servidor Oracle cria este tipo de índice automaticamente quando você define para uma coluna em uma tabela uma constraint PRIMARY KEY ou UNIQUE KEY. O nome do índice é o nome fornecido para a constraint.

O outro tipo de índice que um usuário pode criar é um índice não único (nonunique). Por exemplo, você pode criar um índice em uma coluna definida como FOREIGN KEY para melhorar a performance de um JOIN em uma consulta.

Criando um Índice

Crie um índice em uma ou mais colunas executando o comando CREATE INDEX.

Sintaxe:

```
CREATE [UNIQUE] [BITMAP] INDEX nomeindice ON tabela  
(coluna[, coluna]...);
```

Onde:

nomeindice: é o nome do índice.

tabela: é o nome da tabela.

coluna: é o nome da coluna da tabela a ser indexada.

- Aumente a velocidade de acesso da consulta na coluna NOME da tabela TCLIENTES:
- Crie um índice em uma ou mais colunas:

Exemplo:

```
CREATE INDEX tclientes_nome_idx  
ON tclientes(nome);
```

Diretrizes para a Criação de Índices

Mais nem sempre é Melhor

Muitos índices em uma tabela nem sempre significam melhora na velocidade das consultas. Para cada operação DML que é confirmada em uma tabela os índices devem ser atualizados. Quanto mais índices você associou a uma tabela, mais esforço o Servidor Oracle terá que realizar para atualizar todos os índices após uma operação DML.

Quando Criar um Índice

- A coluna é frequentemente utilizada na cláusula WHERE ou em uma condição de join.
- A coluna possui uma grande faixa de valores.
- A coluna possui um grande número de valores nulos.
- Duas ou mais colunas são frequentemente utilizadas juntas em uma cláusula WHERE ou condição de join.
- A tabela é grande e a maioria das consultas recuperam menos que 2% a 4% das linhas.

Lembre-se que se você quiser garantir unicidade, você deve definir uma constraint do tipo UNIQUE na definição da tabela. Então, um índice único será criado automaticamente.

Quando não Criar um Índice

- A tabela for pequena.
- As colunas não são utilizadas frequentemente como uma condição da consulta.
- A maioria das consultas recuperam mais que 2% a 4% das linhas.
- A tabela é frequentemente atualizada. Se você possui um ou mais índices em uma tabela, os comandos DML que acessam a tabela utilizam relativamente mais tempo para sua execução devido à manutenção dos índices.
- As colunas referenciadas são utilizadas como parte de uma expressão.

Consultando os Índices

```
SELECT ic.index_name, ic.column_name, ic.column_position col_pos,  
       ix.uniqueness  
FROM   user_indexes ix, user_ind_columns ic  
WHERE  ic.index_name = ix.index_name  
AND    ic.table_name = 'TCLIENTES';
```

Confirme a existência de índices a partir da visão do dicionário de dados USER_INDEXES. Você também pode conferir as colunas envolvidas em um índice examinando a visão USER_IND_COLUMNS.

O exemplo acima exibe todos os índices previamente criados, nomes das colunas afetadas e a unicidade para a tabela TCLIENTES.

Removendo um Índice

- Remova um índice do dicionário de dados:

```
DROP INDEX nomeindice;
```

- Remova o índice TCL_NOME_IDX do dicionário de dados:

```
DROP INDEX tclientes_nome_idx;
```

Você não pode modificar índices. Para alterar um índice, você deve removê-lo e então criá-lo novamente. Remova a definição de um índice do dicionário de dados executando o comando DROP INDEX. Para remover um índice, você deve ser o dono do índice ou possuir o privilégio DROP ANY INDEX.

Sintaxe:

nomeindice: é o nome do índice.

Sinônimos

Sintaxe:

```
CREATE [PUBLIC] SYNONYM sinonimo FOR objeto;
```

Para referenciar uma tabela criada por outro usuário, você deve prefixar o nome da tabela com o nome do usuário que a criou seguido por um ponto.

Criando um sinônimo você elimina a necessidade de qualificar o nome do objeto com o schema e o provê um nome alternativo para uma tabela, visão, sequence, procedure ou outros objetos.

Este método pode ser especialmente útil com nomes de objeto longos, como visões.

Onde:

PUBLIC: cria um sinônimo acessível a todos os usuários.

sinônimo: é o nome do sinônimo a ser criado.

objeto: identifica o objeto para o qual o sinônimo deve ser criado.

Diretrizes

- O objeto não pode estar contido em uma package.
- Um nome de sinônimo privado deve ser distinto de todos os outros objetos criados pelo mesmo usuário.

Criando e Removendo Sinônimos

Exemplo:

```
CREATE SYNONYM top
FOR vcontratos_top;
```

O exemplo acima cria um sinônimo privado para a visão VCONTRATOS_TOP para uma referência mais rápida.

O DBA pode criar um sinônimo público acessível para todos os usuários. O exemplo abaixo cria um sinônimo público chamado CLIENTES para a tabela TCLIENTES do usuário DESENV:

Exemplos:

No ambiente de desenvolvimento:

```
CREATE PUBLIC SYNONYM tclientes
FOR desenv.tclientes;
```

No ambiente de produção:

```
CREATE PUBLIC SYNONYM tclientes
FOR prod.tclientes;
```

Removendo um Sinônimo

Para remover um sinônimo, utilize o comando DROP SYNONYM. Somente um DBA pode remover um sinônimo público.

Sintaxe:

```
DROP [PUBLIC] SYNONYM sinonimo;
```

Onde:

sinônimo: é o nome do sinônimo a ser criado.

Exemplos:

```
DROP SYNONYM top;
```

```
DROP PUBLIC SYNONYM tclientes;
```


Exercícios – 14

1. No utilize a sua conexão e conecte-se ao banco de dados do curso. Crie uma sequência para ser utilizada com a coluna da chave primária da tabela PESSOAS. A sequência deve iniciar em 10 e possuir um valor máximo de 200. Forneça para a sequência um incremento de 1 e o nome como SPESSOAS_ID.
2. Crie um arquivo de script para exibir a seguinte informação sobre suas sequências: o nome da sequência, o valor máximo, o incremento e o último número fornecido. Execute o comando.
3. Escreva um script interativo para inserir uma linha na tabela PESSOAS. Coloque o nome do arquivo como *e14q3.sql*. Utilize a sequência que você criou para a coluna ID. Crie um prompt customizado para solicitar o nome da pessoa. Execute o script adicionando duas pessoas chamadas "Maria de Lourdes" e "Samuel Medeiros". Efetive as inserções.
4. Crie um índice para a coluna definida como FOREIGN KEY na tabela CONTRATOS.
5. Mostre os índices que existem no dicionário de dados para a tabela CONTRATOS.

Espaço para anotações

Apêndice 1 – Comandos do SQL*Plus

Objetivos

- Conhecer alguns comandos do com SQL*Plus.

Comandos de Edição do SQL*Plus

Comandos SQL*Plus são inseridos uma linha de cada vez e não são armazenados no buffer:

Comando	Descrição
A[PPEND] <i>text</i>	Adiciona o texto (<i>text</i>) no final da linha corrente
C[HANGE] / <i>old</i> / <i>new</i>	Modifica o texto (<i>old</i>) para o novo (<i>new</i>) na linha corrente
C[HANGE] / <i>text</i> /	Remove o texto (<i>text</i>) da linha corrente
CL[EAR] BUFF[ER]	Remove todas as linhas do SQL buffer
DEL	Remove a linha corrente
I[NPU T]	Insere um número indefinido de linhas
I[NPUT] <i>text</i>	Insere uma linha com o texto (<i>text</i>)
L[IST]	Lista todas as linhas do SQL buffer
L[IST] <i>n</i>	Lista uma linha (especificada por <i>n</i>)
L[IST] <i>m n</i>	Lista um intervalo de linhas (de <i>m</i> até <i>n</i>)
R[UN]	Exibe e executa o comando SQL corrente no buffer
<i>N</i>	Especifica qual linha deve tornar-se a corrente
<i>n text</i>	Substitui a linha (<i>n</i>) com o texto (<i>text</i>)
0 <i>text</i>	Insere uma linha antes da linha 1
CL[EAR] SCR[EEN]	Limpa a tela de exibição do SQL*Plus

Tabela 2-4: Comandos de edição do SQL*Plus

Comandos de Formatação do SQL*Plus

Obtendo Resultados mais Legíveis

Você pode controlar as características do relatório utilizando os seguintes comandos:

Comando	Descrição
COL[UMN] [<i>column option</i>]	Controla a formatação de colunas
TTI[TLE] [<i>text</i> OFF ON]	Especifica um cabeçalho a ser apresentado no topo de cada página
BTI[TLE] [<i>text</i> OFF ON]	Especifica um rodapé a ser apresentado no final de cada página do relatório
BRE[AK] [ON <i>report_element</i>]	Suprime valores duplicados e divide linhas de dados com linhas em branco

Tabela 7-4: Comandos de formatação do SQL*Plus

Diretrizes

- Todos os comandos de formatação permanecem em efeito até o final da sessão do SQL*Plus ou até que o formato fixado seja sobrescrito ou limpo.
- Lembre-se de voltar suas configurações do SQL*Plus para os valores default depois de todo relatório.
- Não existe nenhum comando para configurar uma variável do SQL*Plus para seu valor default; você deve conhecer o valor específico ou encerrar sua sessão e conectar novamente.
- Se você fornecer um alias para sua coluna, você deve referenciar o nome do alias, não o nome da coluna.

Comando COLUMN

```
COL[UMN] [{column|alias} [option]]
```

Opção	Descrição
CLE[AR]	Limpa qualquer formatação de coluna
FOR[MAT] <i>format</i>	Modifica a exibição dos dados de uma coluna
HEA[DING] <i>text</i>	Configura o cabeçalho da coluna. O caractere pipe () pode forçar uma quebra de linha no cabeçalho se você não utilizar justificação
JUS[TIFY] { <i>align</i> }	Justifica o cabeçalho da coluna (não os dados) à esquerda, centralizado ou à direita
NOPRI[NT]	Oculto a coluna
NUL[L] <i>text</i>	Especifica o texto a ser exibido para valores nulos
PRI[NT]	Mostra a coluna
TRU[NCATED]	Trunca a string no final da primeira linha de exibição
WRA[PPED]	Coloca o final da string na próxima linha

Tabela 7-5: Opções do comando COLUMN

Utilizando o Comando COLUMN

- Crie cabeçalhos de coluna:

```
COLUMN nome HEADING 'Curso|Target' FORMAT A15  
COLUMN preco JUSTIFY LEFT FORMAT $99G999D99  
COLUMN pre_requisito FORMAT 999999999 NULL 'Sem Pré-Requisito'
```

- Mostre a configuração atual para a coluna NOME:

```
COLUMN nome
```

```
COLUMN nome CLEAR
```

- Limpe as configurações para a coluna NOME:

Comando	Descrição
COL[UMN] <i>column</i>	Exibe as configurações atuais para a coluna especificada
COL[UMN]	Exibe as configurações atuais para todas as colunas
COL[UMN] column CLE[AR]	Limpa as configurações para a coluna especificada
CLE[AR] COL[UMN]	Limpa as configurações para todas as colunas

Tabela 7-6: Utilizando o comando COLUMN

Se você tiver um comando muito longo, você pode continuá-lo na próxima linha terminando a linha atual com um hífen (-).

Máscaras do Comando COLUMN

Elemento	Descrição	Exemplo	Resultado
An	Exibição com tamanho de n	N/A	N/A
9	Dígito com supressão de zeros	99999	1234
0	Coloca zeros à esquerda	09999	01234
\$	Símbolo de dólar flutuante	\$9999	\$1234
L	Moeda local	L9999	R\$1234
D	Posição da vírgula decimal	9999D99	1234,00
G	Separador de milhar	9G999	1.234

Tabela 7-7: Máscaras do comando COLUMN

A tabela acima exhibe exemplos de formatação de colunas.

O Servidor Oracle exhibe uma string com o caractere (#) no lugar de um número inteiro cujos dígitos excedem o número de dígitos providos pela máscara. Também exhibe uma string com o caractere (#) no lugar de um valor cuja máscara é alfanumérica mas o valor atual é numérico.

Utilizando o Comando BREAK

- Para suprimir duplicidades:

```
SQL> BREAK ON cod_trg ON carga_horaria
```

- Para produzir sumários gerais:

```
SQL> BREAK ON report
```

Utilize o comando BREAK para dividir as linhas e suprimir valores duplicados. Para assegurar que o comando BREAK funcione corretamente, ordene pelas colunas nas quais você está quebrando.

Sintaxe:

```
BREAK on column[|alias|row|report]
```

Limpe todas as configurações de BREAK utilizando o comando CLEAR:

```
CLEAR BREAK
```

Utilizando os Comandos TTITLE e BTITLE

`TTI[TLE] [text|OFF|ON]`

- Configure o cabeçalho do relatório:

`SQL> TTITLE 'Relatório|Cursos'`

- Configure o rodapé do relatório:

`SQL> BTITLE 'Confidencial'`

Utilize o comando TTITLE para formatar cabeçalhos de página e o comando BTITLE para rodapés. Rodapés aparecem ao final de cada página de acordo com o valor de PAGESIZE.

A sintaxe para BTITLE e TTITLE é idêntica. Você pode utilizar o caractere pipe (|) para dividir o texto do título em várias linhas.

Sintaxe:

text: representa o texto de título. Coloque entre aspas simples se o texto for mais de uma palavra.

O exemplo de TTITLE acima configura o cabeçalho do relatório para exibir Relatório centralizado em uma linha e Cursos centralizado na linha seguinte. O exemplo de BTITLE configura o rodapé do relatório para exibir Confidencial. TTITLE automaticamente coloque a data e número da página no relatório.

Criando um Arquivo de Script para Executar um Relatório

Você pode entrar cada um dos comandos do SQL*Plus no prompt de SQL ou colocá-los todos, incluindo o comando SELECT, em um arquivo de comandos (ou script). Um arquivo de script consiste em pelo menos um comando SELECT e vários comandos do SQL*Plus.

Passos para Criar um Arquivo de Script

- Crie o comando SQL SELECT no prompt de SQL. Assegure-se de que os dados necessários para o relatório estejam corretos antes de salvar o comando para um arquivo e aplicar os comandos de formatação. Assegure-se que a classificação da cláusula ORDER BY esteja de acordo com as quebras que você venha a definir para o relatório.
- Salve o comando SELECT para um arquivo de script.
- Edite o arquivo de script para colocar os comandos do SQL*Plus.
- Adicione os comandos de formatação necessários antes do comando SELECT. Assegure-se de não colocar comandos do SQL*Plus dentro do comando SELECT.
- Verifica que o comando SELECT seja seguido por um caractere de execução, ou o caractere (;) ou uma barra (/).
- Adicione comandos para limpar as formatações após o caractere de execução do comando SELECT. Como alternativa, você pode chamar um arquivo padrão que contém todos os comandos de limpeza de formatação.
- Salve o arquivo de script com suas modificações.
- No SQL*Plus, execute o arquivo de script entrando o comando START filename ou @filename. Este comando é necessário para ler e executar o arquivo de script.

Diretrizes

- Você pode incluir linhas em branco entre os comandos do SQL*Plus em um script.
- Você pode abreviar comandos do SQL*Plus.
- Inclua comandos de limpeza de formatação ao término do arquivo para restaurar o ambiente original do SQL*Plus.

Relatório de Exemplo

1	Qua Set 14			page
			Relatório de Cursos	
Código Target	Curso	Horas	Preço	
TAOAS	Administração do Oracle 10G Application Server	18	R\$956,00	
TAT8I	Otimização de Aplicativos Oracle		R\$956,00	
TDR6ID	Oracle 10G Designer: Design e Geração do Banco de Dados		R\$956,00	
TDR6IM	Oracle 10G Designer: Modelagem de Sistemas		R\$956,00	
TDR6IW	Oracle 10G Designer: Design e Geração de Aplicações par		R\$956,00	
TD31A	Oracle 10G Discover para Administradores		R\$956,00	
TD6IREP	Oracle 10G Designer: Design e Geração de Reports		R\$956,00	
TF6I-A	Oracle 10G Developer: Forms - Parte II Avançado		R\$956,00	
TOPC	Oracle 10G Portal - Construindo Portais Corporativos		R\$956,00	
TOPP	Oracle 10G Portal - Construindo Portlets		R\$956,00	
TASPAV	ASP Avançado	20	R\$1.000,00	
TASP00	ASP - Criando uma Loja Virtual em Banco de Dados		R\$1.000,00	
TCOLMX	ColdFusion		R\$1.000,00	
TJFUND	Java Fundamentals		R\$1.000,00	
TOOUML	Orientação a Objetos - Fundamentos UML, Análise e Proje		R\$1.000,00	
TPHPAV	PHP Avançado		R\$1.000,00	
TPHPOO	PHP		R\$1.000,00	
TPOSAD	PostgreSQL: Administração do Banco de dados		R\$1.000,00	
TPOSUP	PostgreSQL: Linguagem Procedural e Unidades de Programa		R\$1.000,00	
TSQLAV	SQL Standard Avançado		R\$1.000,00	
TSQLST	SQL Standard		R\$1.000,00	
TXML00	XML		R\$1.000,00	
TDAWPL	Desenvolvimento de Aplicações Web em PL/SQL	18	R\$1.051,60	
TWAWAS	WebSphere Administração do WebSphere Application Server	30	R\$1.200,00	
TWSADV	WebSphere Studio Application Developer V4.0 - Desenvolv		R\$1.200,00	
WSphPort	WebSphere Portal Server V4.0 - Desenvolvimento de Porta		R\$1.200,00	
TF6I	Oracle 10G Developer: Forms - Parte I		R\$1.275,00	
TR6I	Oracle 10G Developer: Reports		R\$1.275,00	
TJADV	Java Advanced e Acesso a Banco de Dados		R\$1.500,00	
TDR6IG	Oracle 10G Designer: Design e Geração da Aplicação		R\$1.594,00	
TO10G1	Introdução ao Oracle 10G I: Conceitos básicos do Oracle		R\$1.800,00	
	Confidencial			

Espaço para anotações

Apêndice 2 – Soluções dos Exercícios

Soluções Exercícios 2 – Introdução ao comando SELECT utilizando o SPL*PLUS e o Oracle SQL Developer

1. Inicie a sessão do SQL Developer criada neste capítulo.
2. Verifique a existência das tabelas utilizadas no curso.
Utilize o browser no lado esquerdo da tela
3. Verifique a estrutura da tabela TCLIENTES e depois selecione todas as colunas desta tabela.
Clique sobre a tabela TCLIENTES no browse do lado esquerdo da tela

```
SELECT *
FROM tclientes;
```

4. Faça uma query que selecione as colunas nome, preço e desconto da tabela TCURSOS. Coloque o título da coluna DESCONTO como "Desconto Promocional".

```
SELECT nome, preco, desconto "Desconto Promocional"
FROM tcursos;
```

5. Faça um SQL que retorne quais estados possuem clientes.

```
SELECT DISTINCT estado
FROM tclientes;
```

6. Abra uma sessão do SQL*Plus e mostre a estrutura da tabela TITENS.

```
SQL> desc titens
```

7. Existem três erros de codificação neste comando. Você pode os identificar?

```
SELECT id, cod_trg, preço x 0.5 DESCONTO CURSO
FROM tcursos;
```


- A tabela TCURSOS não possui uma coluna chamada PREÇO, o nome correto da coluna correta é PRECO.
 - Operador de multiplicação é "*" e não "x".
 - O alias DESCONTO CURSO possui mais de uma palavra, portanto o alias correto seria DESCONTO_CURSO ou "DESCONTO CURSO".
8. Crie uma consulta para exibir o id, data de compra, desconto e o total para cada contrato, mostrando o id do contrato por primeiro.

```
SELECT id, dt_compra, desconto  
FROM tcontratos;
```

Soluções Exercícios 3 – Restringindo e Ordenando Dados

1. Inicie uma sessão do SQL Developer.
2. Crie uma consulta para exibir o id e a data de compra (dt_compra) dos contratos com o total superior a 10000.

```
SELECT id, dt_compra
FROM tcontratos
WHERE total > 10000;
```

3. Crie uma consulta para exibir o nome, endereço, cidade, cep, telefone, do cliente com o id 140.

```
SELECT nome, endereco, cidade, cep, telefone
FROM tclientes
WHERE id = 140;
```

4. Crie uma consulta para exibir o id, dt_compra, desconto e total dos contratos (tabela tcontratos) para todos os contratos cuja total não está na faixa de 2000 à 5000.

```
SELECT id, dt_compra, desconto, total
FROM tcontratos
WHERE total NOT BETWEEN 2000 AND 5000;
```

5. Altere a consulta para selecionar somente os contratos que possuem desconto maior do que zero. Apresente o resultado em ordem crescente de data de compra.

```
SELECT id, dt_compra, desconto, total
FROM tcontratos
WHERE total NOT BETWEEN 2000 AND 5000
ORDER BY dt_compra;
```

6. Mostre o nome, a cidade e o estado de todos os clientes que possuem estado igual a 'RS' ou 'SP' em ordem alfabética.

```
SELECT nome, cidade, estado
FROM tclientes
WHERE estado IN ('RS', 'SP')
ORDER BY nome;
```

7. Crie uma consulta para exibir os contratos sem desconto.

```
SELECT *  
  FROM tcontratos  
 WHERE desconto is null;
```

Soluções Exercícios 4 – Funções Single Row, Funções de Conversão e Expressões de Condição

1. Conecte-se ao banco de dados com o SQL Developer.
2. Escreva uma consulta para exibir a data atual no formato 'DD/MM/YYYY HH24:MI:SS'. Coloque o alias de coluna como "Data". Execute a consulta.

```
SELECT to_char(sysdate, 'DD/MM/YYYY HH24:MI:SS')  
"Data"  
FROM dual;
```

3. Mostre o id do curso, o código Target (cod_trg), o preço e o preço com um aumento de 15%. Selecione somente os cursos que possuam o número 1 em qualquer parte do id. Coloque o alias da coluna como "Novo Preço". Execute a consulta.

```
SELECT id, cod_trg, preco, preco * 1.15 "Novo  
Preço"  
FROM tcursos  
WHERE TO_CHAR(id) LIKE '%1%';
```

4. Altere a consulta anterior para adicionar uma nova coluna que subtraia o preço antigo do novo preço. Coloque o alias da coluna como "Aumento". Execute a consulta.

```
SELECT id, cod_trg, preco, preco * 1.15 "Novo  
Preço" ,  
       (preco * 1.15) - preco "Aumento"  
FROM tcursos  
WHERE TO_CHAR(id) LIKE '%1%';
```

5. Mostre o nome de cada cliente e calcule o número de meses entre a data atual e a data de nascimento. Coloque o alias da coluna como "Meses de Vida". Ordene o resultado pelo número de meses. Arredonde o número de meses para o número inteiro mais próximo. Execute a consulta.

```
SELECT nome,  
ROUND(MONTHS_BETWEEN(SYSDATE,dt_nascimento))  
      "Meses de Vida"  
FROM tclientes  
ORDER BY 2;
```

6. Crie uma consulta para exibir o código Target e o preço de todos os cursos. Formate o preço para exibir como R\$1.000,00. Coloque o alias da coluna como "Valor Curso". Execute a consulta.

```
SELECT cod_trg, TO_CHAR(preco,'L99G999G999D99')  
      "Valor Curso"  
FROM tcursos;
```

Soluções Exercícios 5 – Exibindo Dados a Partir de Múltiplas Tabelas

1. No utilize a sua conexão e conecte-se ao banco de dados do curso.
2. Escreva uma consulta para exibir o nome do cliente e os contratos (id e data de compra) que este cliente possui, ordene por nome em ordem alfabética. Execute a consulta.

```
SELECT cl.nome, co.id contrato, co.dt_compra
FROM   tclientes cl, tcontratos co
WHERE  cl.id = co.tclientes_id
ORDER BY cl.nome;
```

3. Crie uma lista única de todos os contratos que possuem clientes com a letra 'A' (maiúscula ou minúscula) no nome, ordene por ordem alfabética. Execute a consulta.

```
SELECT cl.nome, co.id contrato, co.desconto,
co.total
FROM   tclientes cl, tcontratos co
WHERE  (cl.id = co.tclientes_id) AND
        UPPER(cl.nome) LIKE '%A%'
ORDER BY cl.nome;
```

4. Escreva uma consulta para exibir o nome do cliente, id do contrato e total, para todos os contratos que não possuem desconto (NULO ou ZERO). Execute a consulta.

```
SELECT  cl.nome, co.id contrato, co.total
FROM    tclientes cl, tcontratos co
WHERE   cl.id = co.tclientes_id
        AND NVL(co.desconto,0) <> 0;
```

5. Crie uma consulta que mostre o nome do cliente, o id e total do contrato, e a classe de desconto para todos os contratos. Ordene pela classe de desconto. Execute a consulta.

```
SELECT cl.nome, co.id, co.total contrato, de.classe
FROM   tclientes cl, tcontratos co, tdescontos de
WHERE  (cl.id = co.tclientes_id) AND
        (NVL(co.desconto,0) BETWEEN de.base_inferior
        AND de.base_superior)

ORDER BY de.classe;
```

6. Mostre o Id, o código e a data de criação do curso e o Id, o código e a data de criação do seu curso pré-requisito do mesmo para todos os cursos. Execute a consulta.

```
SELECT tcs.id "Id do Curso"
      , tcs.cod_trg "Código Curso"
      , tcs.dt_criacao "DT Criação Curso"
      , pre.id "Id do Pré Requisito"
      , pre.cod_trg "Codigo Pré-requisito"
      , tcursos pre
WHERE  tcs.pre_requisito = pre.id;
```

Soluções Exercícios 6 – Utilizando Funções de Grupo e Formando Grupos

1. Funções de grupo atuam sobre muitas linhas para produzir uma única linha para cada grupo formado? **Sim**.
2. Funções de grupo incluem nulos nos cálculos? **Não**.
3. No utilize a sua conexão e conecte-se ao banco de dados do curso. Mostre o maior, o menor, a soma e a média do total de todos os contratos. Coloque o alias das colunas como "Máximo", "Mínimo", "Soma" e "Média". Arredonde os resultados com zero dígitos decimais. Execute a consulta.

```
SELECT  ROUND(MAX(total),0) "Máximo",  
        ROUND(MIN(total),0) "Mínimo",  
        ROUND(SUM(total),0) "Soma",  
        ROUND(AVG(total),0) "Média"  
FROM tcontratos;
```

4. Modifique a consulta anterior para exibir o menor, o maior, a soma e a média do total para cada estado. Execute a consulta.

```
SELECT  estado uf,  
        ROUND(MAX(total),0) "Máximo",  
        ROUND(MIN(total),0) "Mínimo",  
        ROUND(SUM(total),0) "Soma",  
        ROUND(AVG(total),0) "Média"  
FROM tcontratos , tclientes  
WHERE tclientes.id = tcontratos.tclientes_id  
GROUP BY estado;
```

5. Escreva uma consulta que mostre a diferença entre o maior e menor contrato. Coloque o alias da coluna como "DIFERENÇA". Execute a consulta.

```
SELECT MAX(total) - MIN(total) DIFERENÇA  
FROM tcontratos;
```

6. Escreva uma consulta para exibir o estado, o número de contratos e a média do total de contratos para todos os clientes daquele estado. Coloque os alias de coluna como "UF", "CONTRATOS", "MÉDIA", respectivamente. Execute a consulta.


```
SELECT    tcl.estado uf, COUNT(tcn.id) contratos,  
          AVG(tcn.total) média  
FROM      tclientes tcl, tcontratos tcn  
WHERE     tcl.id = tcn.tclientes_id  
GROUP BY  tcl.estado;
```

Soluções Exercícios 7 – Variáveis de Substituição e Variáveis de ambiente do SQL*Plus

1. Conecte-se no SQL*Plus.
2. Escreva um arquivo de script para mostrar o código Target, o preço e a data de criação para todos os cursos que foram criados entre um determinado período. Concatene o código Target e data de criação, separando-os por uma vírgula e espaço, e coloque o alias da coluna como "Curso". Solicite ao usuário os dois intervalos do período utilizando o comando ACCEPT. Utilize o formato DD/MM/YYYY. Salve o script para um arquivo chamado *e7q1.sql*. Execute o script *e7q1.sql*.

```
SET VERIFY OFF
ACCEPT data1 DATE FORMAT 'DD/MM/YYYY' PROMPT
'Informe a 1a Data (DD/MM/YYYY): '
ACCEPT data2 DATE FORMAT 'DD/MM/YYYY' PROMPT
'Informe a 2a Data (DD/MM/YYYY): '
COLUMN cursos FORMAT A25

SELECT cod_trg, to_char(dt_criacao,
'DD/MM/YYYY') data, preco
FROM tcursos
WHERE dt_criacao BETWEEN '&data1' AND '&data2';

UNDEFINE data1
UNDEFINE data2
CLEAR COLUMN
SET VERIFY ON
```

Soluções Exercícios 8 – Sub-consultas

1. No utilize a sua conexão e conecte-se ao banco de dados do curso. Escreva uma consulta para exibir o nome dos clientes e a data de nascimento para todos os clientes que estão no mesmo estado do cliente 'Mário Cardoso', excluindo-o do resultado.

```
SELECT nome, dt_nascimento
FROM tclientes
WHERE estado IN
      (SELECT estado
       FROM tclientes
       WHERE nome = 'Mário Cardoso')
AND nome <> 'Mário Cardoso';
```

2. Crie uma consulta para exibir o id e o total do contrato, o nome do cliente responsável pelo contrato para todos os contratos com total maior que a média do total de contratos. Classifique o resultado em ordem descendente de total. Execute a consulta.

```
SELECT tcn.id, tcn.total, tcl.nome
FROM tclientes tcl, tcontratos tcn
WHERE tcl.id = tcn.tclientes_id
AND tcn.total >
      (SELECT AVG(total)
       FROM tcontratos)
ORDER BY tcn.total DESC;
```

3. Escreva uma consulta que mostre o id e o nome do cliente para todos os clientes que são de um estado com qualquer cliente cujo nome contenha uma letra 'v' (minúscula). Execute a consulta.

```
SELECT id, nome
FROM tclientes
WHERE estado IN
      (SELECT estado
       FROM tclientes
       WHERE nome LIKE '%v%');
```

4. Mostre o código Target e o preço dos cursos com pré-requisito o cod_trg 'THTML4'.

```
SELECT cod_trg, preco
FROM tcursos
WHERE pre_requisito IN
      (SELECT id
        FROM tcursos
        WHERE cod_trg = 'THTML4');
```

5. Mostre o nome e o telefone de todos os clientes que compraram no dia '06/01/2005'. Execute a consulta.

```
SELECT nome, telefone
FROM tclientes
WHERE id IN
      (SELECT tclientes_id
        FROM tcontratos
        WHERE dt_compra = to_date('06/01/2005',
        'DD/MM/YYYY');
```

Soluções Exercícios 9 – Operadores SET

1. No utilize a sua conexão e conecte-se ao banco de dados do curso. Crie uma expressão UNION que represente a união dos totais de cada contrato e das somas dos itens de contrato. No primeiro SELECT selecione o código do contrato, o valor total de todos os contratos e a string fixa definida por 'CONTRATOS'. No segundo SELECT selecione o código do contrato, todas as somas do total de cada item deste contrato e a string fixa 'ITENS'. Ordene pela terceira coluna e depois pela primeira. Defina os alias nos dois SELECTs como ID, TOTAL e STRING. Execute a consulta.

```
SELECT id, total, 'CONTRATOS' STRING
FROM tcontratos
UNION
SELECT tcontratos_id id, SUM(total) total, 'ITENS'
STRING
FROM titens
GROUP BY tcontratos_id
ORDER BY 3,1;
```

2. Crie um script para a operação de subtração (MINUS) entre dois SELECTs e a intersecção (INTERSECT) com um terceiro SELECT. No primeiro SELECT selecione ID, DT_COMPRA e TOTAL de todos os contratos. No segundo SELECT selecione ID, DT_COMPRA e TOTAL de todos os contratos sem desconto (NULO ou ZERO). Subtraia o primeiro pelo segundo. No terceiro SELECT selecione ID, DT_COMPRA e TOTAL dos contratos realizados com clientes de SP. Execute a consulta.

```
SELECT id, dt_compra, total
FROM tcontratos
MINUS
SELECT id, dt_compra, total
FROM tcontratos
WHERE NVL(desconto,0) = 0
INTERSECT
SELECT tcn.id, tcn.dt_compra, tcn.total
FROM tclientes tcl, tcontratos tcn
WHERE tcl.id = tcn.tclientes_id
AND tcl.estado = 'SP';
```

Soluções Exercícios 10 – Manipulando Dados

1. No utilize a sua conexão e conecte-se ao banco de dados do curso. Descreva a estrutura da tabela TCLIENTES_VIP para identificar os nomes das colunas.

Resposta:

```
DESC tclientes_vip
```

2. Adicione a primeira linha de dados na tabela TCLIENTES_VIP a partir do exemplo de dados a seguir.

Resposta:

ID	SOBRENOME	NOME	CLIENTEID	CREDITO
1	Tapes	Carlos	ctapes	795

```
INSERT INTO tclientes_vip
VALUES (1, 'Tapes', 'Carlos', 'ctapes', 795);
```

3. Adicione a segunda linha na tabela TCLIENTES_VIP a partir do exemplo de dados a seguir.

Resposta:

ID	SOBRENOME	NOME	CLIENTEID	CREDITO
2	Moura	Ana	amoura	860

```
INSERT INTO
tclientes_vip(id,sobrenome,nome,clienteid,credito)
VALUES (2, 'Moura', 'Ana', 'amoura', 860);
```

4. Visualize suas inserções para a tabela TCLIENTES_VIP.

Resposta:

```
SELECT *
FROM tclientes_vip;
```

5. Adicione mais três linhas na tabela TCLIENTES_VIP a partir do exemplo de dados a seguir.

Resposta:

ID	SOBRENOME	NOME	CLIENTEID	CREDITO
3	Pinheiro	Viviane	vpinheir	1100
4	Dutra	Manuel	mdutra	750
5	Silva	Cesar	csilva	1550

```

INSERT INTO
tclientes_vip(id,sobrenome,nome,clienteid,credito)
VALUES (3,'Pinheiro','Viviane','vpinheir',1100);

INSERT INTO
tclientes_vip(id,sobrenome,nome,clienteid,credito)
VALUES (4,'Dutra','Manuel','mdutra',750);

INSERT INTO
tclientes_vip(id,sobrenome,nome,clienteid,credito)
VALUES (5,'Silva','Cesar','csilva',1550);

```

6. Visualize suas inserções para a tabela TCLIENTES_VIP.

Resposta:

```

SELECT *
FROM tclientes_vip;

```

7. Torne as inserções permanentes.

Resposta:

```

COMMIT;

```

8. Modifique o sobrenome do cliente com ID = 3 para "Souza".

Resposta:

```

UPDATE tclientes_vip
SET sobrenome = 'Souza'
WHERE id = 3;

```

9. Modifique o crédito para 1000 para todos os clientes com o crédito menor que 900.

Resposta:

```
UPDATE tclientes_vip
SET credito = 1000
WHERE credito < 900;
```

10. Visualize suas modificações para a tabela TCLIENTES_VIP.

Resposta:

```
SELECT *
FROM tclientes_vip;
```

11. Remova o cliente "Ana Moura" da tabela TCLIENTES_VIP.

Resposta:

```
DELETE FROM tclientes_vip
WHERE sobrenome = 'Moura'
AND nome = 'Ana';
```

12. Visualize suas modificações para a tabela.

Resposta:

```
SELECT *
FROM tclientes_vip;
```

13. Efetive todas as modificações pendentes.

Resposta:

```
COMMIT;
```

14. Adicione uma linha para a tabela TCLIENTES_VIP com os valores id = 6, nome = 'Roberto', sobrenome = 'Pires', clienteid = rpires e credito = 2000.

Resposta:

```
INSERT INTO
tclientes_vip(id,sobrenome,nome,clienteid,credito)
VALUES (6,'Pires','Roberto','rpires',2000);
```


15. Visualize sua inserção para a tabela.

Resposta:

```
SELECT *  
FROM tclientes_vip;
```

16. Marque um ponto intermediário no processamento da transação.

Resposta:

```
SAVEPOINT A;
```

17. Remova todas as linhas da tabela TCLIENTES_VIP.

Resposta:

```
DELETE FROM tclientes_vip;
```

18. Visualize que a tabela TCLIENTES_VIP está vazia.

Resposta:

```
SELECT *  
FROM tclientes_vip;
```

19. Descarte a mais recente operação de DELETE sem descartar a operação de INSERT anterior.

Resposta:

```
ROLLBACK TO SAVEPOINT A;
```

20. Visualize a tabela TCLIENTES_VIP.

Resposta:

```
SELECT *  
FROM tclientes_vip;
```

21. Efetive a transação.

Resposta:

```
COMMIT;
```

Soluções Exercícios 11– Criando e Gerenciando Tabelas

1. No utilize a sua conexão e conecte-se ao banco de dados do curso. Crie a tabela PESSOAS baseado na descrição abaixo.

Column Name	Data Type	Nullable
ID	NUMBER(7,0)	Yes
NOME	VARCHAR2(25 BYTE)	Yes

Resposta:

```
CREATE TABLE pessoas
(id NUMBER(7),
nome VARCHAR2(25));
```

2. Insira linhas na tabela PESSOAS com dados a partir das linhas da tabela TCLIENTES. Inclua somente as colunas necessárias (id, nome).

Resposta:

```
INSERT INTO pessoas
SELECT id, nome
FROM tclientes;
```

3. Crie a tabela CONTRATOS baseado na descrição abaixo.

Column Name	Data Type	Nullable
ID	NUMBER(5,0)	Yes
PESSOA_ID	NUMBER(7,0)	Yes
DATA	DATE	Yes
DESCONTO	NUMBER(7,2)	Yes
TOTAL	NUMBER(7,2)	Yes

Resposta:

```
CREATE TABLE contratos
(id NUMBER(5),
Pessoa_id NUMBER(7),
data DATE,
desconto NUMBER(7,2),
total NUMBER(7,2));
```

4. Modifique a coluna ID da tabela CONTRATOS para permitir o uso de valores numéricos de até 7 dígitos (id NUMBER (7)).

Resposta:

```
ALTER TABLE contratos  
MODIFY id NUMBER(7);
```

5. Confirme que as tabelas PESSOAS e CONTRATOS estão armazenadas no dicionário de dados (USER_TABLES).

Resposta:

```
SELECT table_name  
FROM user_tables  
WHERE table_name IN ('PESSOAS', 'CONTRATOS');
```

6. Adicione um comentário para a definição das tabelas PESSOAS e CONTRATOS que as descreva. Confirme sua adição no dicionário de dados.

Resposta:

```
COMMENT ON TABLE pessoas IS 'Informações de  
Pessoas';  
  
COMMENT ON TABLE contratos IS 'Informações de  
Contratos';  
  
SELECT table_name, comments  
FROM user_tab_comments  
WHERE table_name IN ('PESSOAS', 'CONTRATOS');
```

Soluções Exercícios 12 – Implementando Constraints

1. No utilize a sua conexão e conecte-se ao banco de dados do curso. Adicione uma constraint PRIMARY KEY para a tabela CONTRATOS utilizando a coluna ID.

Resposta:

```
ALTER TABLE contratos
ADD CONSTRAINT con_id_pk PRIMARY KEY(id);
```

2. Crie uma constraint PRIMARY KEY na tabela PESSOAS utilizando a coluna ID.

Resposta:

```
ALTER TABLE pessoas
ADD CONSTRAINT pes_id_pk PRIMARY KEY(id);
```

3. Adicione uma referência de chave estrangeira para a tabela CONTRATOS que garanta que o contrato não seja associado a uma pessoa com ID inexistente (coluna PESSOAS_ID).

Resposta:

```
ALTER TABLE contratos
ADD CONSTRAINT con_pes_id_fk FOREIGN
KEY(pessoas_id) REFERENCES pessoas(id);
```

4. Confirme que as constraints foram adicionadas consultando a visão USER_CONSTRAINTS. Observe os tipos e nomes das constraints.

Resposta:

```
SELECT constraint_name, constraint_type
FROM   user_constraints
WHERE  table_name IN ('PESSOAS','CONTRATOS');
```

5. Modifique a tabela CONTRATOS. Adicione uma coluna IMPOSTO com o tipo de dado NUMBER(7,2) definindo que o valor do imposto não pode ser negativo.

Resposta:

```
ALTER TABLE contratos
ADD (imposto NUMBER(7,2) CONSTRAINT
con_imposto_ck CHECK(imposto >0));
```

Soluções Exercícios 13 – Criando Visões

1. No utilize a sua conexão e conecte-se ao banco de dados do curso. Crie uma visão chamada VCONTRATOS baseada no número do contrato, data de compra e no número do cliente a partir da tabela TCONTRATOS. Modifique o cabeçalho para o código do cliente para "CLIENTE".

Resposta:

```
CREATE VIEW vcontratos AS
SELECT id, dt_compra, tclientes_id cliente
FROM tcontratos;
```

2. Mostre o conteúdo da visão VCONTRATOS.

Resposta:

```
SELECT *
FROM vcontratos;
```

3. Selecione a coluna VIEW_NAME e TEXT a partir da visão USER_VIEWS do dicionário de dados.

Resposta:

```
SELECT view_name, text
FROM user_views
WHERE view_name = 'VCONTRATOS';
```

4. Utilizando a visão VCONTRATOS, execute uma consulta para exibir todos os clientes e datas de compras.

Resposta:

```
SELECT cliente, dt_compra
FROM vcontratos;
```

5. Crie uma visão chamada VCLIENTESRS que contenha o número do cliente, o nome, a cidade e o estado para todos os clientes com estado igual a 'RS'. Não permita que operações DML sejam executadas através da visão.

Resposta:

```
CREATE VIEW vclientesrs AS
SELECT id, nome, cidade, estado
FROM tclientes
WHERE estado = 'RS'
WITH READ ONLY;
```

6. Mostre a estrutura e o conteúdo da visão VCLIENTESRS.

Resposta:

```
DESCRIBE vclientesrs

SELECT *
FROM vclientesrs;
```

Soluções Exercícios 14 – Outros Objetos do Banco de Dados

1. No utilize a sua conexão e conecte-se ao banco de dados do curso. Crie uma sequência para ser utilizada com a coluna da chave primária da tabela PESSOAS. A sequência deve iniciar em 10 e possuir um valor máximo de 200. Forneça para a sequência um incremento de 1 e o nome como SPESSOAS_ID.

Resposta:

```
CREATE SEQUENCE spessoas_id
START WITH 10
INCREMENT BY 1
MAXVALUE 200;
```

2. Crie um arquivo de script para exibir a seguinte informação sobre suas sequências: o nome da sequência, o valor máximo, o incremento e o último número fornecido. Coloque o nome do arquivo como *e14q2.sql*. Execute o script criado.

Resposta:

```
SELECT sequence_name, max_value, increment_by,
last_number
FROM user_sequences;
```

3. Escreva um script interativo para inserir uma linha na tabela PESSOAS. Coloque o nome do arquivo como *e14q3.sql*. Utilize a sequência que você criou para a coluna ID. Crie um prompt customizado para solicitar o nome da pessoa. Execute o script adicionando duas pessoas chamadas "Maria de Lourdes" e "Samuel Medeiros". Efetive as inserções.

Resposta:

```
ACCEPT pname PROMPT 'Informe o nome da pessoa: '

INSERT INTO pessoas(id,nome)
VALUES (spessoas_id.NEXTVAL, '&pnome');

UNDEFINE pname

COMMIT;
```

4. Crie um índice para a coluna definida como FOREIGN KEY na tabela CONTRATOS.

Resposta:

```
CREATE INDEX con_pessoa_idx  
ON contratos (pessoas_id);
```

5. Mostre os índices que existem no dicionário de dados para a tabela CONTRATOS. Salve o comando para um script chamado *e14q5.sql*.

Resposta:

```
SELECT index_name, table_name, uniqueness  
FROM user_indexes  
WHERE table_name = 'CONTRATOS';
```