Banco de Dados Avançado

Prof.ª Ms. Simone Cristina Aléssio

UNIASSELVI
2015

Elaboração:
Prof.ª Ms. Simone Cristina Aléssio

Revisão, Diagramação e Produção:
Centro Universitário Leonardo da Vinci – UNIASSELVI

Ficha catalográfica elaborada na fonte pela Biblioteca Dante Alighieri
UNIASSELVI – Indaial.

005.75

A372b Aléssio, Simone Cristina

Banco de dados avançado / Simone Cristina Aléssio. Indaial : UNIASSELVI, 2015.

205 p. : il.

ISBN 978-85-7830-878-0

1. Banco de dados específicos.
I. Centro Universitário Leonardo Da Vinci.

Impresso por:

APRESENTAÇÃO

Prezado(a) Acadêmico(a)!

Bem-vindo(a) à disciplina BANCO DE DADOS AVANÇADO. Este Caderno de Estudos foi elaborado com o intuito de contribuir e aprimorar o seu desenvolvimento acerca da prática de desenvolvimento de *software* em Banco de Dados.

O conteúdo está dividido em três unidades. Na Unidade 1, você vai encontrar uma breve revisão teórica acerca de conceitos de Sistemas de Informação, processo de desenvolvimento de *software*, Banco de Dados e modelagem de dados.

A Unidade 2 concentra a linguagem SQL com seus tipos de dados, além dos comandos DDL e DML. Nesta unidade também são elaborados algoritmos em Banco de Dados, explorando-se profundamente a construção de selects e cursores, com a utilização de comandos de manipulação, funções matemáticas e de agrupamento.

A Unidade 3 apresenta uma visão geral acerca dos objetos de Banco de Dados que comportam a programação na base: procedures, funções, *packages* e *triggers*. Este Caderno pressupõe alguma experiência anterior em programação, não fazendo parte de nossos objetivos o ensino de programação básica. Pretendemos fazer com que você pegue os conhecimentos adquiridos nas disciplinas de algoritmos e programação básica e passe a utilizá-los em combinação com os conceitos da programação em Banco de Dados, na resolução de problemas. Isso não quer dizer que você precisa ser um *expert* em programação para acompanhar este conteúdo.

Para facilitar seu estudo e desenvolvimento das rotinas, todos os comandos possuem exemplos que você poderá facilmente compreender e executar na base de dados.

Aproveitamos esse momento para destacar que os exercícios **NÃO SÃO OPCIONAIS**. O objetivo de cada exercício deste Caderno é a fixação de determinado conceito através da prática. É aí que reside a importância da realização de todos. Sugerimos fortemente que, em caso de dúvida em algum exercício, você entre em contato com seu tutor externo ou com a tutoria da Uniasselvi e que não passe para o exercício seguinte enquanto o atual não estiver completamente compreendido.

Bom estudo!
Prof.ª Simone Cristina Aléssio

Você já me conhece das outras disciplinas? Não? É calouro? Enfim, tanto para você que está chegando agora à UNIASSELVI quanto para você que já é veterano, há novidades em nosso material.

Na Educação a Distância, o livro impresso, entregue a todos os acadêmicos desde 2005, é o material base da disciplina. A partir de 2017, nossos livros estão de visual novo, com um formato mais prático, que cabe na bolsa e facilita a leitura.

O conteúdo continua na íntegra, mas a estrutura interna foi aperfeiçoada com nova diagramação no texto, aproveitando ao máximo o espaço da página, o que também contribui para diminuir a extração de árvores para produção de folhas de papel, por exemplo.

Assim, a UNIASSELVI, preocupando-se com o impacto de nossas ações sobre o ambiente, apresenta também este livro no formato digital. Assim, você, acadêmico, tem a possibilidade de estudá-lo com versatilidade nas telas do celular, *tablet* ou computador.

Eu mesmo, UNI, ganhei um novo *layout*, você me verá frequentemente e surgirei para apresentar dicas de vídeos e outras fontes de conhecimento que complementam o assunto em questão.

Todos esses ajustes foram pensados a partir de relatos que recebemos nas pesquisas institucionais sobre os materiais impressos, para que você, nossa maior prioridade, possa continuar seus estudos com um material de qualidade.

Aproveito o momento para convidá-lo para um bate-papo sobre o Exame Nacional de Desempenho de Estudantes – ENADE.

Bons estudos!

BATE PAPO SOBRE O ENADE!

Olá, acadêmico!

Você já ouviu falar sobre o **ENADE**?

Se ainda não ouviu falar nada sobre o ENADE, agora você receberá algumas informações sobre o tema.

Ouviu falar? Ótimo, este informativo reforçará o que você já sabe e poderá lhe trazer novidades.

Vamos lá!

Qual é o significado da expressão ENADE?

EXAME NACIONAL DE DESEMPENHO DOS ESTUDANTES

Em algum momento de sua vida acadêmica você precisará fazer a prova ENADE.

Que prova é essa?

É **obrigatória**, organizada pelo INEP – Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira.

Quem determina que esta prova é obrigatória... O **MEC – Ministério da Educação**.

O objetivo do MEC com esta prova é o de avaliar seu desempenho acadêmico assim como a qualidade do seu curso.

Fique atento! Quem não participa da prova fica impedido de se formar e não pode retirar o diploma de conclusão do curso até regularizar sua situação junto ao MEC.

Não se preocupe porque a partir de hoje nós estaremos auxiliando você nesta caminhada.

Você receberá outros informativos como este, complementando as orientações e esclarecendo suas dúvidas.

Você tem uma trilha de aprendizagem do ENADE, receberá e-mails, SMS, seu tutor e os profissionais do polo também estarão orientados.

Participará de webconferências entre outras tantas atividades para que esteja preparado para #mandar bem na prova ENADE.

Nós aqui no NEAD e também a equipe no polo estamos com você para vencermos este desafio.

Conte sempre com a gente, para juntos mandarmos bem no ENADE!

UNIASSELVI

SUMÁRIO

UNIDADE 1

REVISÃO TEÓRICA E MODELAGEM DE DADOS

OBJETIVOS DE APRENDIZAGEM

Ao final desta unidade você será capaz de:

- identificar e caracterizar os principais tipos de Sistemas de Informação;
- relembrar os principais conceitos de Banco de Dados e Sistemas Gerenciadores de Banco de Dados;
- recapitular os conceitos de modelagens de dados, com enfoque para a modelagem relacional.

PLANO DE ESTUDOS

Esta unidade de ensino está dividida em 3 tópicos, sendo que no final de cada um deles você encontrará atividades que contribuirão para a apropriação dos conteúdos.

TÓPICO 1 – SISTEMAS DE INFORMAÇÃO

TÓPICO 2 – DESENVOLVIMENTO DE *SOFTWARE*

TÓPICO 3 – MODELAGEM DE DADOS

TÓPICO 1

SISTEMAS DE INFORMAÇÃO

1 INTRODUÇÃO

Sistemas de Informação são utilizados em organizações para várias atividades de planejamento, controle, comunicação e organização. Um sistema de informação pode existir sem computador, porém, hoje em dia, é comum associar o conceito com termos técnicos que remetem a *software*, *hardware* e redes. Com esta associação, temos o que se chama de sistema de informação computadorizado ou "automatizado". Um exemplo disso é um sistema de consultório médico, que armazena as informações do paciente, seu histórico de saúde, medicamentos, exames, consultas etc. (BERGAMASCHI, 2004).

Ainda de acordo com o autor, por muito tempo os Sistemas de Informação existiram sem os computadores. Durante séculos, eram compostos somente por pessoas, papéis e tinta para a escrita e documentação das informações. Na sequência, surgiram máquinas de escrever e de cálculos aritméticos, originando os primeiros Sistemas de Informação para o comércio daquela época.

Pode-se entender Sistema como um conjunto de partes interagentes e interdependentes que, conjuntamente, formam um todo unitário com determinado objetivo e efetuam determinada função (DATE, 2002).

Sistema (em computação): uma coleção de homens, máquinas e métodos organizados para realizar um conjunto de funções específicas.

Sistemas de Informação (SI): conjunto de componentes inter-relacionados que coletam, processam, armazenam e distribuem informações para apoiar o controle e a tomada de decisão em uma organização.

Um Sistema de Informação contém informações sobre uma organização e seu ambiente. Além do suporte à tomada de decisão, coordenação, controle, auxilia gerentes e funcionários a analisar problemas, visualizar soluções e também criar novos produtos e serviços.

2 DADOS, INFORMAÇÃO E CONHECIMENTO

Para fixarmos melhor o conceito de Sistemas de Informação, precisamos entender o que é a **informação** propriamente dita: dado processado, guardado ou transmitido. Conhecimento amplo resultante da análise e combinação de vários dados (STEINBUHLER, 2004).

A definição acima agrega dois conceitos importantes: dado e conhecimento.

> Dados são apenas os símbolos que usamos para representar a informação, o registro de diferentes aspectos de um fato ou fenômeno. Os números que guardamos em um banco de dados são, como diz o nome, 'dados'. Dados não são interpretados, eles existem, são adquiridos de alguma forma, via coleta, pesquisa ou criação, guardados de outra forma e, possivelmente, apresentados em uma terceira. O computador é uma máquina que manipula dados (LAUDON; LAUDON, 2009, p. 65).

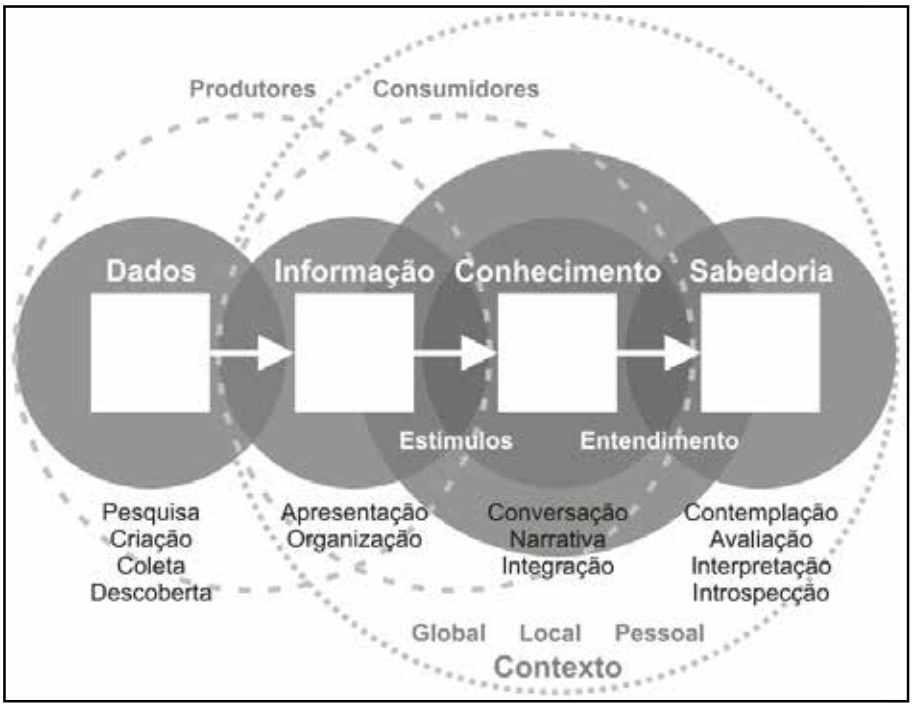
Ainda de acordo com o autor, podemos entender que a informação é um conjunto de dados com significado e determinada utilidade e aplicabilidade. Por exemplo: sempre que encontramos respostas para perguntas como "quando, onde, quanto, quem, como, qual" temos explicitamente uma informação na composição da resposta. Logo, a informação é um dado, ou um conjunto de dados que apresenta um significado.

Não é raro termos que diferenciar dados de informações em Sistemas de Informação. Em muitos sistemas é preciso ter a informação do tipo de pessoa: física ou jurídica. Normalmente, isto é gravado no Banco de Dados através de um domínio numérico com os valores 0 e 1. Neste caso, os números 0 e 1 são os dados que fazem referência à informação do tipo de pessoa.

E quando temos o **conhecimento** então? Sempre que conseguimos aplicar a informação. Toda vez que conseguirmos responder à pergunta "Como?", possivelmente estaremos diante de conhecimento, pois a resposta requer argumentos, descrições, explicações e justificativas (SHEDROFF, 1999).

Ainda de acordo com o autor, compreensão e sabedoria complementam o conceito de conhecimento. A compreensão permite responder a questões baseadas no "por que". Já a sabedoria pode ser entendida como um processo único e individual, cuja capacidade de compreensão e entendimento dos fatos não podem ser compartilhados.

FIGURA 1 - DADOS, INFORMAÇÃO E CONHECIMENTO



FONTE: Shedroff (1999, p. 271)

3 CARACTERISTICAS DOS SISTEMAS DE INFORMAÇÃO

Duas características importantes se destacam quando analisamos os Sistemas de Informação: Interatividade e Reatividade. Sistemas de Informação são altamente interativos e reativos. São interativos, porque trocam informações com o ambiente, com pessoas e outros sistemas computadorizados. Reativos, porque têm reações distintas se sofrerem alterações bruscas em seu ambiente (REYNOLDS, 2002).

A maioria dos Sistemas de Informação atuais também são sistemas de respostas planejadas, uma vez que podemos criar programas para produzi-las. Isso significa que todos os questionamentos feitos ao sistema são definidos, modelados e identificados previamente (O´BRIEN, 2007).

Podemos citar como características atuais dos SI:

- suportam grande volume de informações;
- complexidade de processamentos;
- muitos clientes e/ou usuários envolvidos;
- contexto abrangente, mutável e dinâmico;
- interligação de diversas técnicas e tecnologias;
- suporte à tomada de decisões empresariais;
- auxílio na qualidade, produtividade e competitividade organizacional.

O maior objetivo dos SI é auxiliar o processo de tomada de decisão nos três níveis organizacionais: operacional, gerencial e estratégico. Um bom SI deve ter foco para o principal negócio empresarial, não para negócios secundários ou de apoio para a organização. Devem ser objetivos, ergonômicos, de fácil utilização e entendimento. Do contrário, sua existência é irrelevante (REYNOLDS, 2002).

4 OS SISTEMAS DE INFORMAÇÃO E A ORGANIZAÇÃO

Os Sistemas de Informação atuais atendem a todos os setores organizacionais e são indispensáveis para que eles se mantenham competitivos em seus respectivos mercados de atuação (STAIR, 1998).

Os níveis organizacionais atendidos pelos sistemas e seus respectivos responsáveis são definidos por Laudon e Laudon (2001), conforme detalhado a seguir:

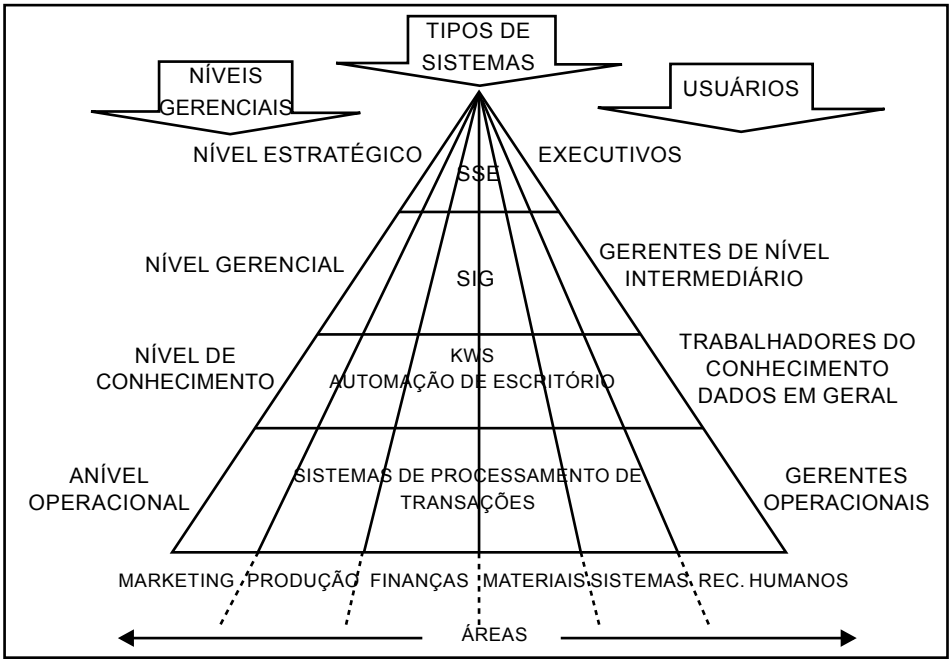
Sistemas de nível operacional, que tratam da execução, acompanhamento e registro da operação diária da empresa, sendo geralmente sistemas fortemente transacionais. Exemplos: sistemas de vendas, folha de pagamento etc.

Sistemas de nível de conhecimento, que suportam as pessoas que trabalham com dados e conhecimento dentro da organização. Exemplos simples de sistemas desse tipo são os processadores de texto e as planilhas eletrônicas.

Sistemas de nível gerencial, que utilizam dados da operação e outros dados inseridos nesses sistemas para permitir a obtenção de informações que permitam a gerência da empresa, suportando a tomada de decisões, o controle e o monitoramento.

Sistemas de nível estratégico, que são sistemas destinados a decisões de mais alto nível (efeito estratégico) e utilizam dados de todos os sistemas anteriores, normalmente de forma agregada e processada, sendo utilizados pela alta gerência.

FIGURA 2 - NÍVEIS DOS SISTEMAS DE INFORMAÇÃO DENTRO DE UMA ORGANIZAÇÃO



FONTE: Laudon e Laudon (2009)

Ainda de acordo com os autores, a cada nível de Sistemas de Informação podemos associar um ou mais tipos de Sistemas de Informação:

Sistemas de Suporte Executivo (SSE), encontrados no nível estratégico. Apoiam executivos do alto escalão da organização na definição de estratégia a prazo. Utilizam-se de dados fortemente internos e externos à organização. Fazem projeções através da simulação de cenários. Tem interface interativa.

Sistemas de Apoio à Decisão (SAD), encontrados no nível gerencial. São utilizados em todos os níveis gerenciais, através da extração de relatórios e estatísticas previamente preparados para possibilitar uma segura análise de dados.

Sistemas de Informação Gerencial (SIG), também encontrados no nível gerencial, são utilizados pelos vários níveis de gerência. Utilizam grande volume de dados ou sumários de transações e modelos simples para obter relatórios sumários (agregados) e de exceções.

Sistemas de Trabalho com Conhecimento (STC), encontrados no nível de conhecimento, utilizam projetos, especificações e bases de conhecimento em geral para produzir modelos e gráficos. Normalmente, são utilizados por profissionais com nível superior.

Sistemas de Escritório (SE), encontrados no nível de conhecimento, tem como objetivo aumentar a produtividade na manipulação de dados em um escritório. Permitem a manipulação de documentos, correio eletrônico e agendas.

Sistemas Processamento de Transações (SPT), encontrados no nível operacional, tratam eventos e transações e fornecem relatórios detalhados, listas e sumários, utilizados pelos gerentes, além de documentos específicos para a transação em que são utilizados.

Os SPT dão suporte a todas as operações organizacionais, além de originar os dados que posteriormente serão utilizados pelos demais Sistemas de Informação.

Você sabia que 95% dos Sistemas de Informação existentes no mundo utilizam algum tipo de Banco de Dados?

Sistemas de Informação eficientes impactam profundamente nas estratégias corporativas e na competitividade das organizações, trazendo benefícios para elas, para seus clientes internos e externos, para os usuários e demais envolvidos em seu manuseio.

Entre os benefícios que as organizações obtêm através dos Sistemas de Informações, podemos destacar:

- suporte à tomada de decisão profícua;
- valor agregado ao produto (bens e serviços);
- melhor serviço e vantagens competitivas;
- produtos de melhor qualidade;
- oportunidade de negócios e aumento da rentabilidade;
- mais segurança nas informações, menos erros, mais precisão;
- aperfeiçoamento nos sistemas, eficiência, eficácia, efetividade, produtividade;
- carga de trabalho reduzida;
- redução de custos e desperdícios;
- controle das operações.

5 ERPS

Um conceito amplamente divulgado atualmente é o ERP, que são Sistemas de Gestão Empresarial – na prática –, isto é, são sistemas de planejamento ou de recursos que integram e centralizam as atividades da empresa através um Banco de Dados único. O líder mundial do mercado é a SAP AG, com o produto SAP R/3. O custo de implantação de um ERP de grande porte pode chegar até 300 milhões de dólares.

Sistemas ERP são modularizados, ou seja, contemplam um conjunto de sistemas de informações necessário à gestão organizacional, como: ERP atuais contêm módulos representando os mais típicos sistemas de informações necessários em uma empresa, tais como: Contabilidade Fiscal e Gerencial, Contas a Pagar e Receber, Controle de Estoque, Produção, RH, Relacionamento com o Cliente, Importações, Exportações, Fretes, Controle de Pedidos e Vendas, entre tantos outros.

RESUMO DO TÓPICO 1

Neste tópico, você aprendeu:

- Os principais tipos de sistemas de informação e como eles apoiam as diversas necessidades de informação e a tomada de decisão de gerentes.

- Que o tipo de informações requeridas pelos gerentes está diretamente relacionado com o nível de gerenciamento e o grau de estrutura nas situações de decisão que eles enfrentam.

- Sistemas de informação gerencial: concentram-se em fornecer aos gerentes produtos de informações pré- especificadas que relatem o desempenho da organização. Os SIG destinam-se a apoiar de forma indireta os tipos mais estruturados de decisões envolvidos no planejamento e controle operacional e tático. Objetivo do SIG: Fornecer informações sobre o desempenho das funções e processos organizacionais básicos, tais como marketing, fabricação e finanças.

- Sistemas de Informação Executiva: os Sistemas de Informação Executiva (EIS) são sistemas de informação que combinam muitas características dos sistemas de informação gerencial e dos sistemas de apoio à decisão. Os EIS se concentram em atender as necessidades de informações estratégicas da alta administração. A meta dos EIS é fornecer aos altos executivos acesso fácil e imediato a informações sobre os fatores críticos ao sucesso (CSFs) de uma empresa, ou seja, os fatores-chave decisivos para a realização dos objetivos estratégicos de uma organização.

- Sistemas de Apoio à Decisão: os sistemas de apoio à decisão são uma das principais categorias de sistemas de informação gerencial. São sistemas de informação computadorizados que fornecem aos gerentes apoio interativo de informações durante o processo de tomada de decisão.

- Sistemas de Processamento de Transações: os SPTs monitoram, coletam, armazenam e processam dados gerados em todas as transações da empresa. Esses dados são a entrada para o Banco de Dados da organização.

AUTOATIVIDADE

1) Escolha um tipo de negócio de pequeno porte, como uma agência de viagens, e descubra (ou imagine) quais são os principais Sistemas de Informação que ela necessita ou pode usar.

2) Classifique os sistemas anteriores quanto ao seu nível na organização.

3) Classifique os sistemas anteriores quanto ao seu tipo.

4) Imagine que esse negócio se torna um grande negócio, por exemplo, uma grande cadeia de agências de viagens, e descubra (ou imagine) que novos sistemas podem ser necessários.

5) Que sistemas de informação fazem parte de seu dia a dia? Que papel você assume ao utilizar esses sistemas?

6) De que Sistemas de Informação você pode se lembrar que contêm informações importantes sobre sua vida pessoal ou profissional?

7) Imagine uma empresa de plano de saúde que possui um sistema de nível operacional que registra e permite a aprovação pela pessoa responsável de exames e consultas. Que Sistemas de Informação de outros níveis podem ser feitos para utilizar essa informação?

8) Que outros Sistemas de Informação podem fornecer informação para o sistema de aprovação?

9) Defina, para um Sistema de Informação, cujas informações necessárias foram escolhidas por você, que dados as descrevem e que conhecimento pode ser obtido a partir delas.

10) Identifique e descreva as razões pelas quais os Sistemas de Informação são, nos dias de hoje, tão importantes para as empresas.

11) Por qual razão algumas empresas obtêm maior valor dos seus Sistemas de Informação do que outras?

TÓPICO 2

DESENVOLVIMENTO DE *SOFTWARE*

1 INTRODUÇÃO

É importante termos em mente que os Sistemas de Informação nascem do Desenvolvimento de *Software*, que podemos definir como uma atividade desenvolvida em curto, médio ou longo prazo, com o intuito de criar um ou mais programas de computador, para atender às necessidades de uma pessoa específica ou uma organização de forma geral.

Observe que são várias as atividades que compõem o desenvolvimento de *software* e que podem ser agrupadas de quatro formas distintas:

- Atividades de Análise, cuja finalidade é descobrir "o que" deve ser feito.
- Atividades de Projeto, cuja finalidade é descobrir "como" o *software* deve ser feito.
- Atividades de Implementação, cuja finalidade é produzir o produto de *software* de acordo com as especificações produzidas nas fases anteriores.
- Atividades de Controle de Qualidade, nas quais se incluem todas as atividades, com o objetivo de garantir a qualidade do produto, como testes e verificações.

Dependendo do processo de desenvolvimento escolhido, as atividades listadas acima podem ser divididas em outras microatividades, que podem ser executadas de formas distintas e por equipes diferentes, inclusive

2 ANÁLISE DE SISTEMAS

A etapa da análise do sistema consiste em descrever os requisitos técnicos e funcionais do sistema de forma a atender à necessidade do usuário. Na verdade, específica o que o sistema deverá fazer e como deverá funcionar, que tarefas deverá executar. Nesta etapa define-se tudo o que faz parte do escopo do sistema, ou seja, delimita-se o cenário que será construído em termos de programação e interfaces.

Segundo Pressman (2006, p. 35),

Todos os métodos de análise devem ser capazes de suportar cinco atividades:
· Representar e entender o domínio da informação.
· Definir as funções que o *software* deve executar.
· Representar o comportamento do *software* em função dos eventos externos.
· Separar os modelos de informação, função e comportamento de maneira a apresentar os detalhes de forma hierárquica.
· Prover a informação essencial em direção à determinação dos detalhes de implementação.

Através da análise de sistemas é possível documentar e comunicar as informações dos sistemas em construção.

3 MODELOS DE ANÁLISE

- Modelo de Negócio: descreve como funciona o negócio onde o sistema está inserido.
- Modelo de Dados: descreve os dados guardados pela memória do sistema na forma de um Modelo Conceitual e segundo o método de entidades e relacionamentos.
- Modelo Funcional: descreve a funcionalidade essencial do sistema, utilizando diagramas de fluxo de dados.

4 O ANALISTA DE SISTEMAS

O Analista de Sistemas desempenha um papel importante na construção dos Sistemas de Informação, pois ele será o responsável pelo levantamento das necessidades junto aos usuários, transformando-as em especificações técnicas e funcionais para a equipe de desenvolvimento.

Um bom analista de sistemas deve ter entendimento do negócio, características precisas, saber ouvir e questionar, investigar, organizar, fazer diagnósticos, prever situações, avaliar, simplificar situações. Estas características são importantes, uma vez que ele desempenhará vários papéis no ciclo de vida do sistema, não se restringindo somente à função de análise de sistemas.

5 CICLO DE VIDA E PROCESSO DE DESENVOLVIMENTO

O ciclo de vida dos Sistemas de Informação compreende três aspectos importantes: concepção, crescimento e morte, que se dividem entre as fases:

a) Concepção: é o projeto de sistema que tem origem em uma ideia ou necessidade, ou ainda, em melhorias de sistemas já existentes.
b) Construção: contempla a análise de dados, especificações de funcionalidades, construção da programação, das interfaces.

c) Implantação: entrega do sistema ao usuário.
d) Implementações: processo de melhorias ou correções de erros durante ou após a implantação.
e) Maturidade: o nível de maturidade está relacionado ao extremo uso do sistema e à satisfação do usuário em relação às suas expectativas.
f) Declínio: dificuldade de continuidade, impossibilidade de agregação de funções necessárias, insatisfação do cliente e/ou usuários.
g) Manutenção: elaboração de manutenções, por exigência legal ou correção de erros, visando à tentativa de sobrevivência do sistema.
h) Morte: descontinuidade do Sistema de Informação.

RESUMO DO TÓPICO 2

Neste tópico, você aprendeu:

- Que um processo de desenvolvimento de *software* pode ser visto como um conjunto de atividades organizadas, usadas para definir, desenvolver, testar e manter um *software.*

- Que o Ciclo de Vida do Desenvolvimento de Sistemas, conhecido também como o "ciclo de vida do *software*", refere-se aos estágios de concepção, projeto, criação e implementação de um SI.

- Que os desenvolvedores de sistemas usam dados, processos e modelos para compreender os sistemas existentes e projetar os novos. Estes modelos fornecem uma linguagem que os analistas, os projetistas e os desenvolvedores podem usar para comunicar-se eficientemente. Os administradores de equipes de desenvolvimento de sistemas se beneficiarão da compreensão destes modelos de modo que possam melhor comunicar suas necessidades.

AUTOATIVIDADE

1) Dê exemplos de empresas que usam os Sistemas de Informação nos seguintes itens:
a) Competição:
b) Criatividade e imaginação na solução de problemas:
c) Aspectos internacionais:
d) Tecnologia e avanços:

2) Em sua opinião, como os Sistemas de Informação podem ajudar nos negócios empresariais?

3) O que significa a frase: "Estamos na era da informação"?

4) Descreva um Sistema de Informação com seu respectivo ciclo de vida.

TÓPICO 3

MODELAGEM DE DADOS

1 INTRODUÇÃO

Bancos de Dados é uma coleção de dados inter-relacionados e persistentes que representam um subconjunto dos fatos presentes em um domínio de aplicação (universo de discurso). São operados pelos Sistemas Gerenciadores de Bancos de Dados (SGBD), que surgiram na década de 70. Antes destes, as aplicações usavam sistemas de arquivos do sistema operacional para armazenar suas informações (SILBERSCHATZ; KORTH, 2004).

Vantagens do uso de Banco de Dados:

- Os dados são armazenados em um único local, evitando-se a redundância deles.
- Os dados são compartilhados por diversas aplicações, facilitando a integração e evitando redefinições.
- Novas operações de manipulação de dados não requerem modificação "pesada" no código da aplicação.
- As aplicações não se preocupam mais com o gerenciamento dos dados.
- Maior flexibilidade de acesso.
- Praticidade das linguagens de programação para BD.

2 TRANSAÇÕES COM BANCO DE DADOS

Podemos considerar as transações como um conjunto de procedimentos executados pelo Banco de Dados e que são imperceptíveis pelo usuário. A integridade de uma transação depende de quatro propriedades, conhecidas como ACID, onde:

Atomicidade

Uma operação atômica é toda transação que não pode ser executada pela metade, deve ser completa. Exemplo: ajuste no saldo de uma conta bancária através de rotinas de programação em Banco de Dados (DATE, 2002).

FIGURA 3 - ATORES DO PROCESSO DE MODELAGEM DE DADOS

FONTE: A autora

Consistência

As restrições impostas pelos Banco de Dados devem ser seguidas e obedecidas integralmente para garantir a veracidade das operações e informações através de regras impostas pelas chaves primárias, estrangeiras, por domínios e campos restritos (HEUSER, 2004).

Isolamento

Cada transação deve ser isolada dos efeitos da execução concorrente de outras transações. Deve garantir que operações executadas simultaneamente tenham o mesmo resultado se executadas de forma serial (DATE, 2002).

Durabilidade

Os resultados das operações são permanentes e só podem ser desfeitos por transações específicas. Exemplo: todos os dados e status relativos a uma transação devem ser armazenados num repositório permanente, não sendo passíveis de falha por uma falha de hardware (SILBERSCHATZ; KORTH, 2004).

3 CONTROLE DE CONCORRÊNCIA EM BANCO DE DADOS

Todo Banco de Dados utilizado por mais de um usuário deverá tratar a concorrência de acesso às informações para garantir a performance e integridade delas. Para isso, o SGBD deverá garantir que todas as transações finalizadas com sucesso não sejam perdidas ou abortadas por operações concorrentes (HEUSER, 2004).

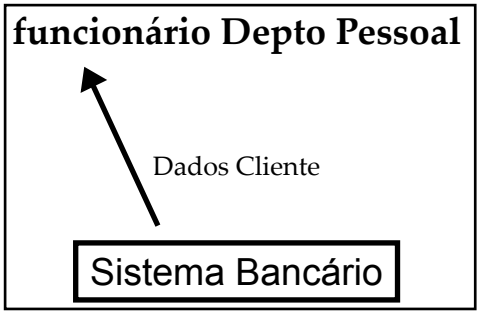
Para o autor, os SGBD utilizam-se de técnicas para garantir a consistência dos dados. A mais utilizada é a serialização, que é a execução de operações uma atrás da outra, sendo necessário saber o início e fim de cada operação. Como suporte ao procedimento, o banco utiliza as regras ACID (Atomicidade, Consistência, Isolamento e Durabilidade).

Leia mais em: Controle de concorrência entre transações em bancos de dados. Disponível em: <http://www.devmedia.com.br/controle-de-concorrencia-entre-transacoes-em-bancos-de-dados/27756#ixzz3SPNd61jc>.

4 SEGURANÇA EM BANCO DE DADOS

Nem todos os usuários dos Bancos de Dados têm autorização para acessar todos os dados (DATE, 2002).

FIGURA 4 - ATORES DO PROCESSO DE MODELAGEM DE DADOS



FONTE: A autora

Os Bancos de Dados SQL implementam mecanismos que restringem ou permitem acessos aos dados de acordo com papéis ou roles fornecidos pelo administrador. O comando GRANT concede privilégios específicos para um objeto (tabela, visão, banco de dados, função, linguagem procedural, esquema ou espaço de tabelas) para um ou mais usuários ou grupos de usuários (SILBERSCHATZ; KORTH, 2004).

5 SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS

Sistema Gerenciador de Banco de Dados: conjunto de programas responsável pelo gerenciamento dos dados em um BD. É o componente crítico em qualquer estrutura de TI. Todos os sistemas de informações e, consequentemente, a maioria das operações da empresa dependem dele (HEUSER, 2004).

Características do SGBD:

Independência de Dados: capacidade de modificar a definição dos esquemas em determinado nível, sem afetar o esquema do nível superior.

Independência de dados física: modifica o esquema físico sem que, com isso, qualquer programa aplicativo precise ser modificado.

Independência de dados lógica: modifica o modelo lógico sem que, com isso, as aplicações precisem ser modificadas.

Principais atores: Administrador do Banco de Dados; Analista ou Projetista da Base de Dados; Programador e Usuário Final.

Alguns Componentes Funcionais de um SGBD:

Compilador DML: traduz os comandos DML da linguagem de consulta em instruções de baixo nível.

Componentes para o tratamento de consultas: executam as instruções de baixo nível geradas pelo compilador DML.

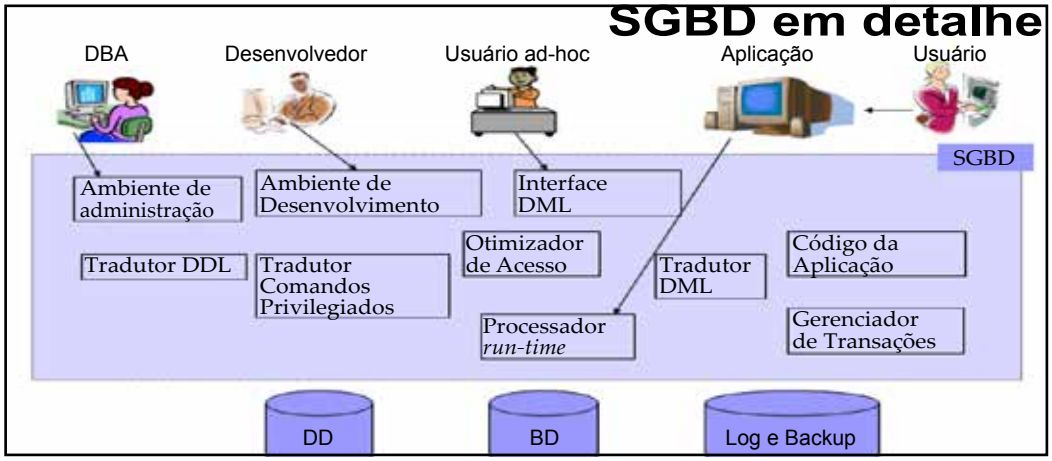
Interpretador DDL: converte comandos DDL em um conjunto de tabelas contendo metade dos que são armazenados no Dicionário de Dados.

Gerenciador de autorizações e integridade: testa o cumprimento das regras de integridade e a permissão ao usuário no acesso ao dado.

Gerenciador de transações: garante a consistência dos dados, independente de falhas do sistema.

Gerenciador de arquivos: gerencia a alocação de espaço de armazenamento e estruturas de dados usadas para representação.

FIGURA 4 - ATORES DO PROCESSO DE MODELAGEM DE DADOS



FONTE: A autora

Um sistema de gerenciamento de Banco de Dados implica a criação e manutenção de bases de dados, elimina a necessidade de especificação de definição de dados, age como interface entre os programas de aplicação e os ficheiros de dados físicos e separa as visões lógica e de concepção dos dados. Assim sendo, são basicamente três os componentes de um SGBD:

Linguagem de definição de dados (especifica conteúdos, estrutura a base de dados e define os elementos de dados).

Linguagem de manipulação de dados (para poder alterar os dados na base).

Dicionário de dados (guarda definições de elementos de dados e respectivas características – descreve os dados, quem os acede etc. [questões de informação]). (DATE, 2002).

Um modelo de SGBD define como os dados serão armazenados no Banco de Dados. Os quatro modelos mais conhecidos são: hierárquico; em rede; relacional; orientado a objetos.

Existem também outros modelos, variando com o autor:

- O modelo de dados objeto-relacional é praticamente uma mistura do modelo relacional com o orientado a objetos.

- O modelo relacional estendido é uma adição de características do modelo orientado a objetos ao relacional.

- O semiestruturado é dedicado a documentos em formatos semiestruturados, normalmente em XML.

- Estruturas de dados otimizadas que possam manipular uma grande quantidade de informação.

- Uma linguagem que possibilite a criação, atualização e consulta dos dados armazenados. Normalmente, esta linguagem é dividida em partes:

Linguagem de definição de dados ou LDD (ou DDL, do inglês), com comandos como CREATE, DROP e ALTER TABLE.

Linguagem de manipulação de dados, ou LMD (ou DML, do inglês), com comandos como UPDATE, SELECT, INSERT e DELETE.

Linguagem de controle de dados, ou LCD, com comandos para controle de acesso dos usuários do sistema, como GRANT e REVOKE em SQL.

- Um mecanismo transacional que garanta a consistência, entre as operações, dos dados armazenados.

Também é possível definir uma linguagem adicional para restrições, como a OCL. As principais linguagens para manipular Bancos de Dados são: SQL, em seus vários padrões, como SQL2 e SQL3.

Exemplo de SGBDS:

IBM, Informix, PostgreSQL, Firebird, HSQLDB, DB2, mSQL, MySQL, Oracle,SQL-Server, TinySQL, ZODB, JADE, Sybase, entre outros.

6 MODELAGEM DE DADOS

O processo de modelagem consiste em criar um modelo para definir e explicar características de funcionamento de um *software*. A modelagem de dados facilita o entendimento do projeto, reduzindo erros de construção na programação e interfaces. Especifica regras de negócios e a estrutura geral do Banco de Dados. É o desenho do sistema de informações, com enfoque para as entidades e seus relacionamentos (HEUSER, 2004).

Podemos utilizar três modelos na construção de um Sistema de Informação:

Modelagem Conceitual: é usada como representação de alto nível e considera exclusivamente o ponto de vista do usuário criador dos dados.

Modelagem Lógica: concentra detalhes da construção.

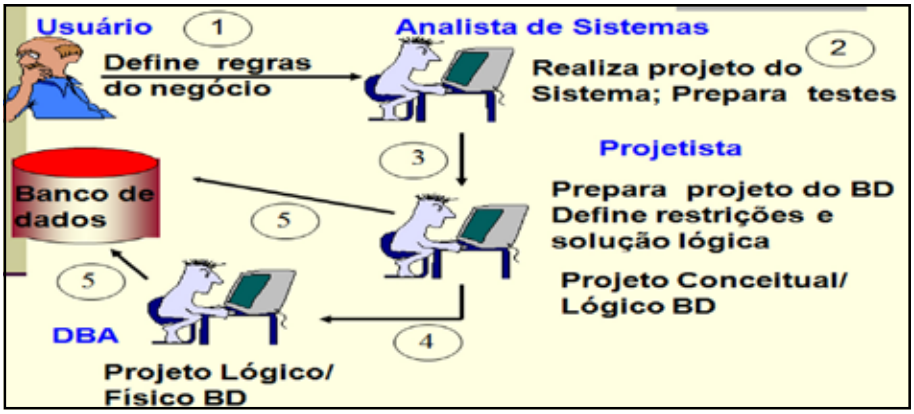
Modelagem Física: demonstra como os dados são fisicamente armazenados na base de dados.

Quanto ao objetivo, podemos identificar as seguintes variações:

- modelagem de dados entidade-relacionamento (leitura, construção e validaçãodos modelos);
- modelagem de relacionamentos complexos, grupos de dados lógicos e ciclo de vida das entidades;
- modelagem de dados corporativa;
- modelagem de dados distribuídos (cliente/servidor);
- modelagem e reengenharia de dados legados;
- modelagem de dados para Data Warehouse.

7 ATORES DO PROCESSO DE MODELAGEM

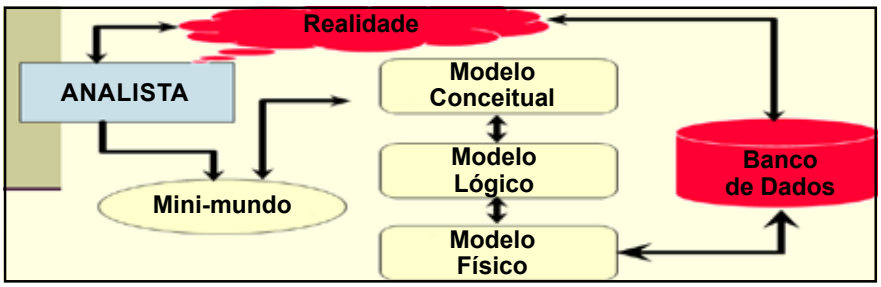
FIGURA 6 - ATORES DO PROCESSO DE MODELAGEM DE DADOS



FONTE: A autora

7.1 NÍVEL DE ABSTRAÇÃO DO PROCESSO DE MODELAGEM

FIGURA 7 - ABSTRAÇÃO DA MODELAGEM DE DADOS



FONTE: A autora

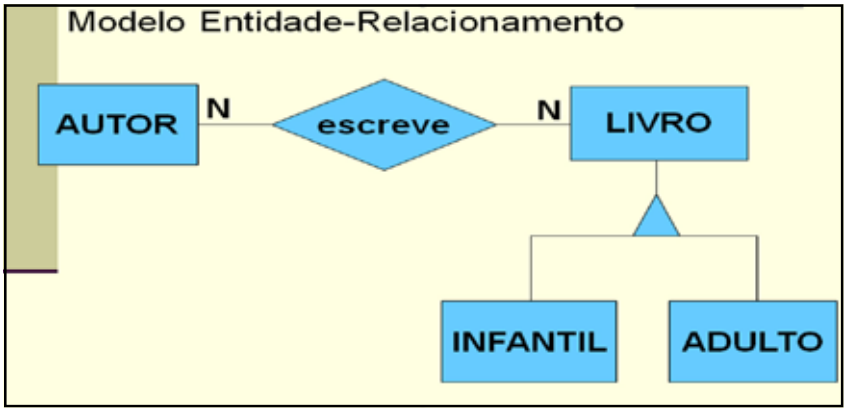
8 MODELO CONCEITUAL DE DADOS

O Modelo Conceitual de Dados descreve as informações contidas no sistema. Pode ser elaborado na forma de fluxograma, tendo como objetivo a criação do Banco de Dados do sistema, intermediado por sistemas de gerenciamento de Banco de Dados. O modelo deve conter as informações primordiais para a execução do sistema, apresentando um detalhamento de suas funcionalidades. Define as tabelas e os relacionamentos que serão armazenados na base de dados. Estabelece e delimita o escopo de trabalho do sistema que será construído (HEUSER, 2004).

Segundo o autor, vale lembrar que o modelo conceitual permite que se identifiquem os resultados das operações dos sistemas sem se importar como estas operações são executadas. Normalmente, descreve um ambiente em observação, abordando sempre a visão do usuário final. O modelo conceitual é independente da tecnologia que será utilizada na construção do sistema.

Exemplo de modelo conceitual:

FIGURA 8 - EXEMPLO DE DIAGRAMA ER



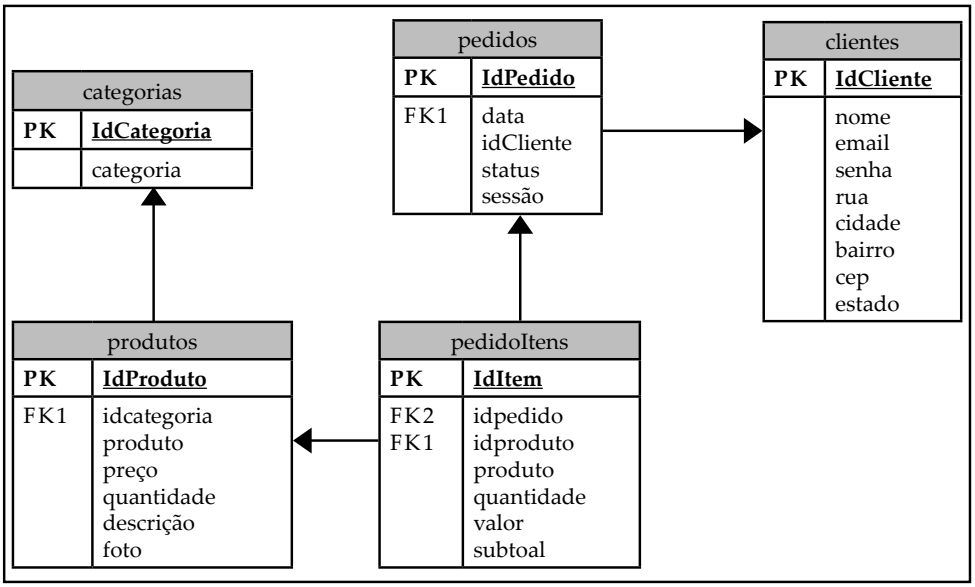
FONTE: A autora

Após a definição do modelo conceitual, inicia-se o desenvolvimento do modelo lógico de dados, que está diretamente relacionado com a tecnologia que será utilizada na construção da programação e interfaces para o usuário.

No modelo lógico são definidas as chaves primárias e secundárias. Este modelo utiliza as regras de normalização para aplicar a integridade referencial nas entidades envolvidas (DATE, 2002).

➔ Exemplo: Tabela/Relação -> Modelo Relacional

FIGURA 9 - EXEMPLO DO MODELO LÓGICO DE DADOS



FONTE: A autora

Após a construção do modelo lógico, a visão que deve prevalecer é a do desenvolvedor e que vai constituir o modelo físico de dados, sendo utilizado na implementação do sistema (linguagens de programação, SGDB, sistema operacional e *hardware*). O modelo físico partirá do lógico e descreverá as estruturas físicas de armazenamento de dados (tamanho de campos, índices, métodos de acesso do SGBD etc.).

De forma geral, o objetivo dos modelos consiste em:

Comunicação com clientes:

- Pode-se mostrar ao cliente, através de um produto de demonstração, parte ou todo o comportamento externo de um sistema.

Visualização:

- Permite visualizar ideias antes de torná-las concretas.

Redução da complexidade:

- Os modelos reduzem a complexidade dividindo-a em um pequeno número de coisas importantes a serem tratadas de cada vez.

9 MODELO ENTIDADE-RELACIONAMENTO

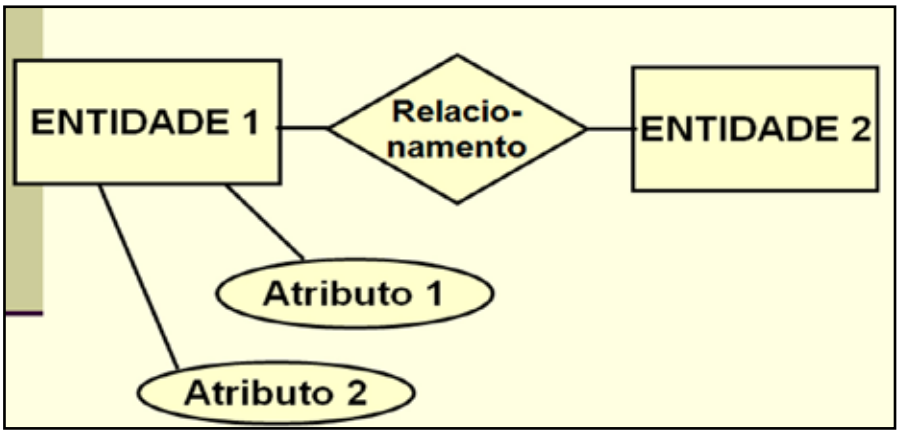
Histórico

Em 1976, Peter P. Chen publicou "**The Entity-Relationship Model**: **Toward the unified view of data**", tendo como base a Teoria Relacional (CODD, 1970).

É, atualmente, o modelo mais utilizado no desenvolvimento de *softwares* em Banco de Dados. É o modelo que se apresenta mais próximo da visão do usuário, sendo utilizado em todo o projeto do BD (HEUSER, 2004).

É através deste modelo que são modelados as entidades e os relacionamentos, permitindo a especificação de um esquema que represente a estrutura lógica geral do Banco de Dados. Produz um diagrama fácil de ser entendido pelo usuário final (COUGO, 1997).

FIGURA 10 - ENTIDADES, ATRIBUTOS E RELACIONAMENTOS



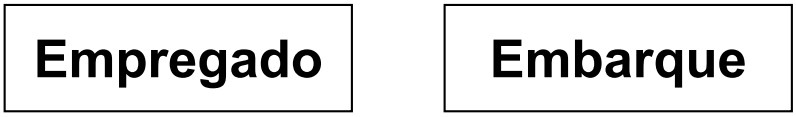
FONTE: A autora

10 CONCEITOS DA ABORDAGEM ENTIDADE RELACIONAMENTO

10.1 ENTIDADES

Representam qualquer "coisa" (concreta ou abstrata) sobre o qual se deseja manter informações, ou seja, são um conjunto de objetos do mundo real sobre os quais se deseja armazenar informações no Banco de Dados (DATE, 2002). Ex.: Empregado, empresa, consulta, embarque.

São representadas graficamente por um retângulo. Pode ser interpretada como uma tabela de dados, onde cada linha representa uma instância.



10.1.1 Atributos de uma entidade

São características, valores descritos, propriedades ou dados associados a uma entidade ou relacionamento. Em outras palavras, descrevem as características da entidade. Ex.: são atributos da entidade Pessoa: nome, endereço, telefone etc. (COUGO, 1997).

Para cada atributo há um conjunto de valores permissíveis (domínio).

FIGURA 11 - EXEMPLO DE ENTIDADE

Entidade: Automóvel					
Placa	**Marca**	**Chassi**	**Proprietário**	**Fabricante**	**Ano**
JHK-3456	Gol	3KG00324MH9	José Batista	Volkswagen	2001
MSN-3289	Fiesta	5GH00845MH8	Carla Maia	Ford	2002
JHG-5634	Meriva	7JK00887MH8	Ana Gomes	Chevrolet	2002

FONTE: A autora

10.2 RELACIONAMENTOS

É a ligação que ocorre entre as tabelas, através dos atributos que são chave primária em uma tabela e chave estrangeira ou FK na outra. Ou seja, são relações associações existentes entre entidades (COUGO, 1997).

Pessoa possui automóvel.
Professor ensina aluno.
Pessoa mora em apartamento.
Francisco é casado com Maria.
Marcos passeia em um barco.
Carla pilota avião.
Documento pertence ao processo.

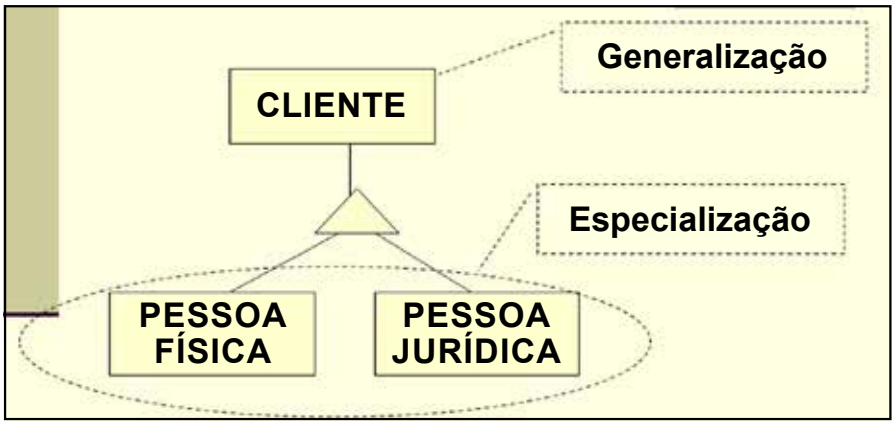
Generalização:

Resultado da **união** de dois ou mais conjuntos de entidades de nível mais baixo, produzindo um conjunto de entidades de nível mais alto (SILBERSCHATZ; KORTH, 2004).

Especialização:

Resultado da **separação** de um subconjunto de entidades de nível mais alto, formando um conjunto de entidades de nível mais baixo (DATE, 2002).

FIGURA 12 - TIPOS DE ENTIDADES



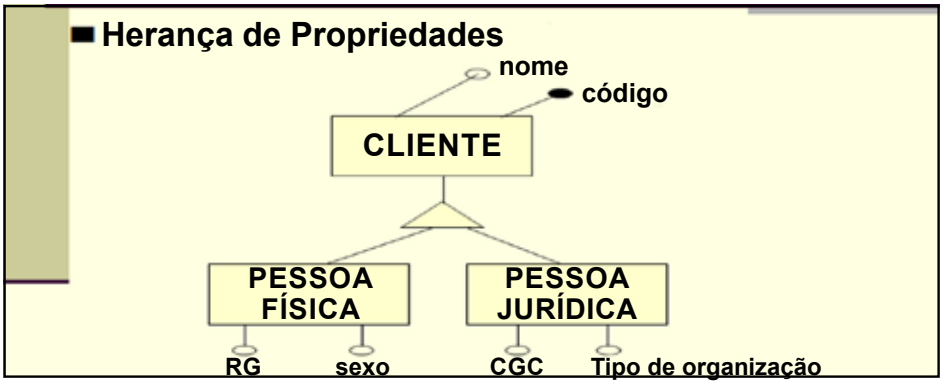
FONTE: A autora

Herança de Propriedades

Ainda de acordo com Date (2002), cada instância da entidade especializada possui, além de suas próprias propriedades, também as propriedades da instância da entidade genérica correspondente.

- ➔ Atributos.
- ➔ Relacionamentos.
- ➔ Generalizações ou especializações.

FIGURA 13 - HERANÇA EM ENTIDADES

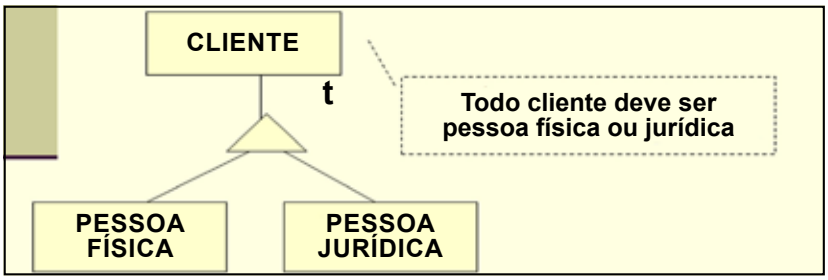


FONTE: A autora

Herança Total

Segundo Date (2002), para cada instância da entidade genérica existe sempre uma instância em uma das entidades especializadas.

FIGURA 14 - Entidade especialista

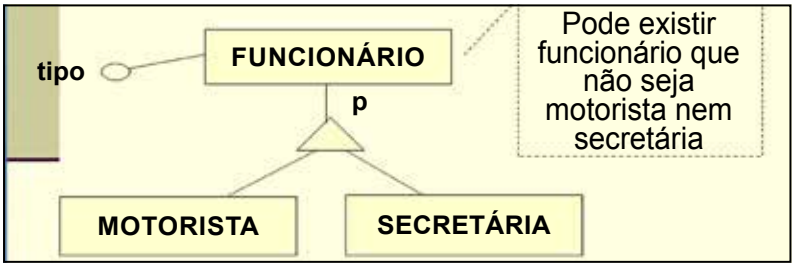


FONTE: A autora

Herança Parcial

Nem toda ocorrência da entidade genérica possui correspondente em entidade especializada (COUGO, 1997).

FIGURA 15 - HERANÇA PARCIAL



FONTE: A autora

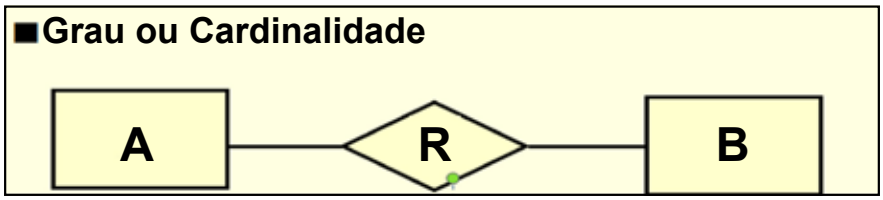
11 SOBRE OS RELACIONAMENTOS

Grau ou Cardinalidade

É o número de ocorrências de uma entidade, que podem estar associadas a uma ocorrência de outra entidade em um relacionamento (HEUSER,2004).

1. Com quantos elementos de B se relaciona *cada um* dos elementos de A?
2. Dado um elemento de B, com quantos elementos de A ele se relaciona?

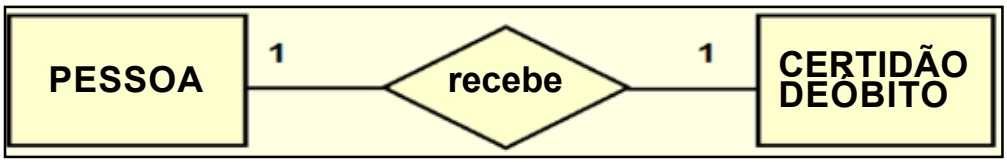
FIGURA 16 - CARDINALIDADE



FONTE: A autora

RELACIONAMENTOS DE 1 PARA 1 (1 : 1)

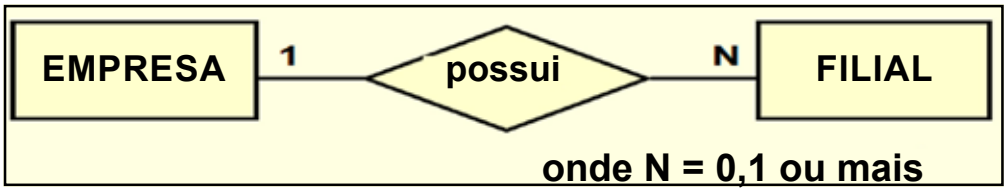
FIGURA 17 - RELACIONAMENTO 1 : 1



FONTE: A autora

RELACIONAMENTOS DE 1 PARA N (1 : N)

FIGURA 18 - RELACIONAMENTO 1 : N



FONTE: A autora

RELACIONAMENTOS N PARA N (N : N)

FIGURA 19 - RELACIONAMENTO N : N

FONTE: A autora

Observe o relacionamento a seguir:

FIGURA 20 - EXEMPLO DE RELACIONAMENTO

FONTE: A autora

A figura acima é o relacionamento entre as entidades **EMPREGADO DEPENDENTE**. Considere as seguintes questões:

Um empregado pode não ter dependentes?

Um dependente pode ter mais de um empregado associado?

Determinado empregado pode possuir mais de um dependente?

Pode existir dependente sem algum empregado associado?

Para responder aos questionamentos acima, é necessário definir uma propriedade importante dele – sua **cardinalidade**.

Podemos extrair do relacionamento a cardinalidade **mínima** e **máxima**. A cardinalidade máxima expressa o número máximo de ocorrências de determinada entidade, e a cardinalidade mínima expressa o número mínimo de ocorrências de determinada entidade associada a uma ocorrência da entidade em questão através do relacionamento. A seguinte convenção é utilizada para a extração das duas cardinalidades:

Número (Mínimo, Máximo)

Para fazermos a leitura do modelo, partimos de determinada entidade e a cardinalidade correspondente a essa entidade é representada no lado oposto. Em nosso exemplo, a cardinalidade **(0:N)** faz referência ao **EMPREGADO**, já a cardinalidade **(1:1)** faz referência ao **DEPENDENTE**. Isso significa que:

- Uma ocorrência de empregado pode não estar associada a uma ocorrência de dependente ou pode estar associada a várias ocorrências dele (determinado empregado pode não possuir dependentes ou pode possuir vários).
- Uma ocorrência de dependente está associada a apenas uma ocorrência de empregado (determinado dependente possui apenas um empregado responsável).

12 NORMALIZAÇÃO

A normalização de dados é uma série de passos que se seguem no projeto de um Banco de Dados, a fim de evitar que ocorram problemas no projeto a ser desenvolvido, bem como a redundância (repetição) de informações, permitindo, desta forma, um armazenamento consistente e um eficiente acesso aos dados em Bancos de Dados relacionais. A normalização não permite a mistura de assuntos em uma mesma tabela. Por exemplo: Não é permitido atribuir informações de um cliente em uma tabela de Pedidos (BATISTA, 2015).

O objetivo da normalização de dados é obter um projeto de BD que elimine a redundância de dados. Desta forma, objetivos básicos dos SGBD são alcançados:

- Independência de dados.
- Consistência de dados.

Para que o SGBD garanta essas características, depende do projetista informar de maneira adequada como os dados estarão dispostos e relacionados no BD.

O processo de normalização aplica uma série de regras sobre as tabelas para aferir se elas foram corretamente criadas na base de dados. Embora a literatura aborde a existência de cinco formas normais (5 FN), na prática, utilizam-se praticamente 3 FN (SANTOS, 2014).

Geralmente, após a aplicação das formas normais, uma tabela pode dar origem a outras tabelas, aumentando o número previsto na modelagem inicial. Isto é necessário para simplificar e tornar mais objetivos os atributos característicos de uma determinada tabela, evitando futuras manutenções nos códigos de programação e também nas interfaces (SILBERSCHATZ; KORTH, 2004).

O maior objetivo do processo de normalização é garantir que todas as tabelas estejam pelo menos na terceira forma normal. A quarta e quinta formas normais são difíceis de serem aplicadas em softwares comerciais (SANTOS, 2014).

A primeira forma normal (ou 1FN) requer que todos os valores de colunas em uma tabela sejam atômicos (ex.: um número é um átomo, enquanto uma lista ou um conjunto não o são). A normalização para a primeira forma normal elimina grupos repetidos, pondo-os cada um em uma tabela separada, conectando-os com uma chave primária ou estrangeira (SILBERSCHATZ; KORTH, 2004).

Processo:

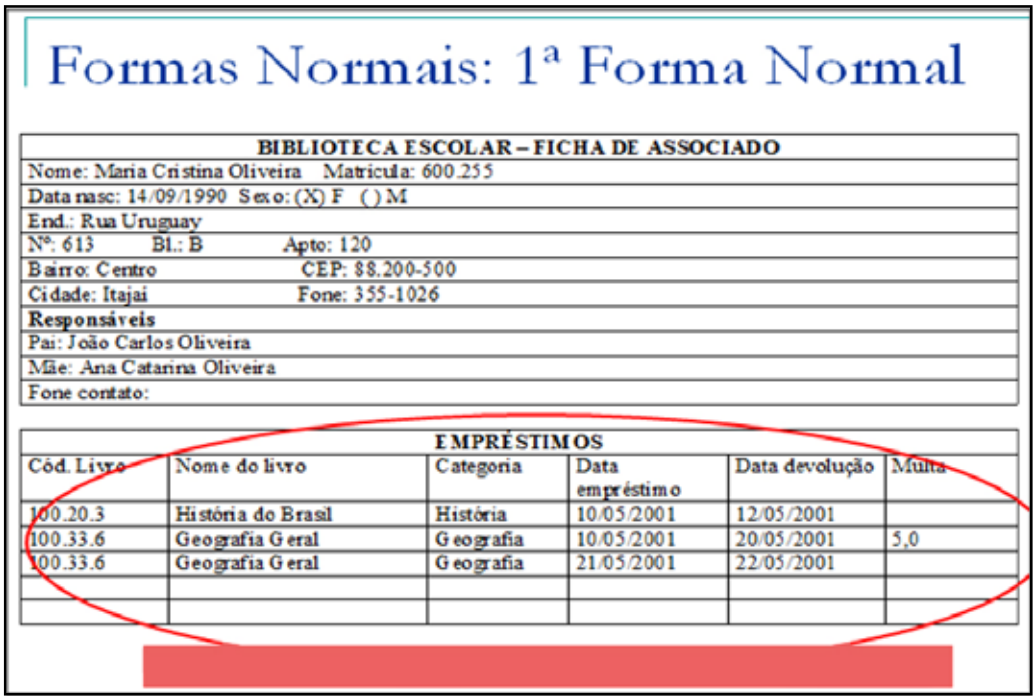
1. Definir as chaves candidatas e escolher a chave primária da tabela.
2. Transformar atributos compostos em atômicos.
3. Em cada tabela eliminar grupos repetitivos, gerando novas linhas, uma para cada ocorrência de item repetitivo, mantendo os valores dos demais itens.

Uma tabela está na 1FN quando não contém tabelas aninhadas

Uma relação está na 1FN quando todos os seus atributos são atômicos e monovalorados.

Em cada tabela eliminar grupos repetitivos, gerando novas linhas, uma para cada ocorrência de item repetitivo, mantendo os valores dos demais itens.

FIGURA 21 - TABELA NORMALIZAÇÃO



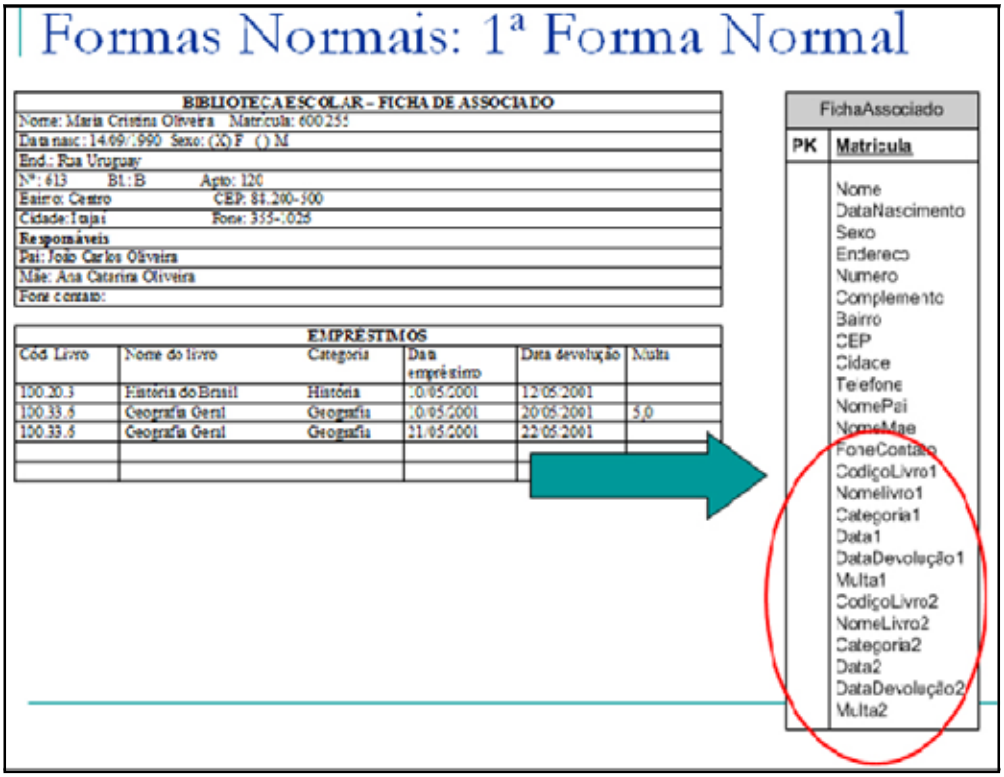
Formas Normais: 1ª Forma Normal

BIBLIOTECA ESCOLAR – FICHA DE ASSOCIADO					
Nome: Maria Cristina Oliveira Matrícula: 600.255					
Data nasc: 14/09/1990 Sexo: (X) F ( ) M					
End.: Rua Uruguay					
Nº: 613 Bl.: B Apto: 120					
Bairro: Centro CEP: 88.200-500					
Cidade: Itajaí Fone: 355-1026					
Responsáveis					
Pai: João Carlos Oliveira					
Mãe: Ana Catarina Oliveira					
Fone contato:					

EMPRÉSTIMOS					
Cód. Livro	Nome do livro	Categoria	Data empréstimo	Data devolução	Multa
100.20.3	História do Brasil	História	10/05/2001	12/05/2001	
100.33.6	Geografia Geral	Geografia	10/05/2001	20/05/2001	5,0
100.33.6	Geografia Geral	Geografia	21/05/2001	22/05/2001	

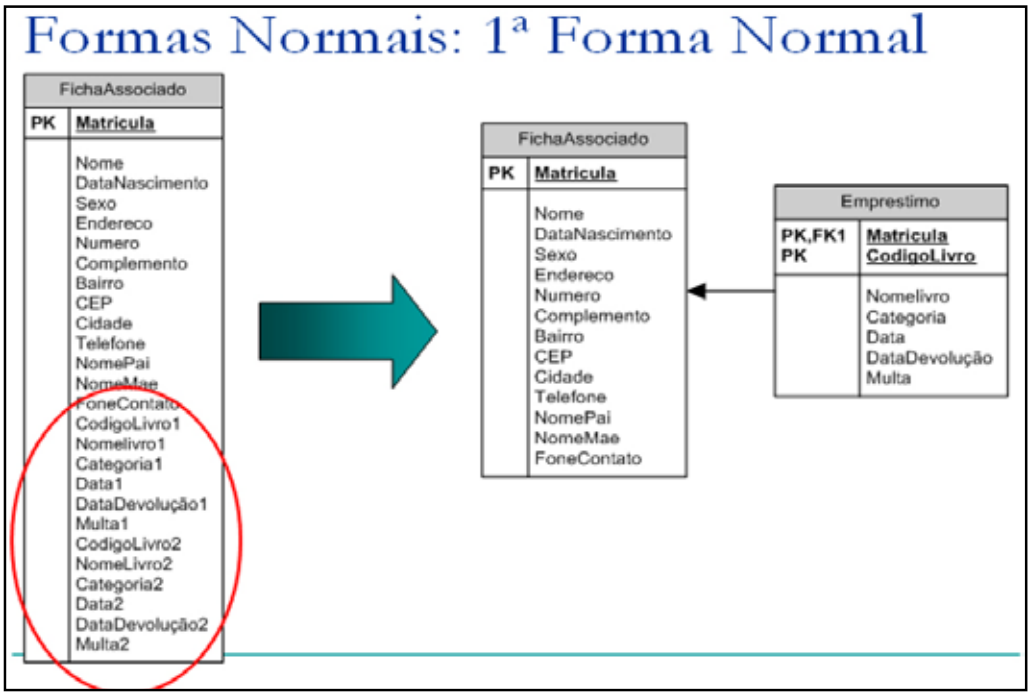
FONTE: A autora

FIGURA 22 - TABELA NORMALIZAÇÃO



FONTE: A autora

FIGURA 23 - TABELA NORMALIZAÇÃO



FONTE: A autora

Uma tabela está na 2FN quando, **além de estar na 1FN**, não contém dependências funcionais parciais. Dependências funcionais parciais: ocorre quando uma coluna depende apenas de parte de uma chave primária composta (SANTOS, 2014).

Já está na 2FN a tabela que possui PK de apenas uma coluna e possui apenas colunas PK.

Evita:
Inconsistência devido à duplicidade de informações
Perda de dados em operações de remoções/alteração na relação

Processo:

1. Identificar as colunas que não fazem parte da chave primária da tabela.
2. Para cada uma das colunas identificadas, analisar se seu valor é determinado por parte, ou pela totalidade da chave.
3. Para as colunas dependentes parcialmente da chave:
    - Criar novas tabelas onde a chave primária será a coluna da chave primária original que determinou o valor da coluna analisada.
    - Excluir da tabela original as colunas dependentes parcialmente da chave primária.

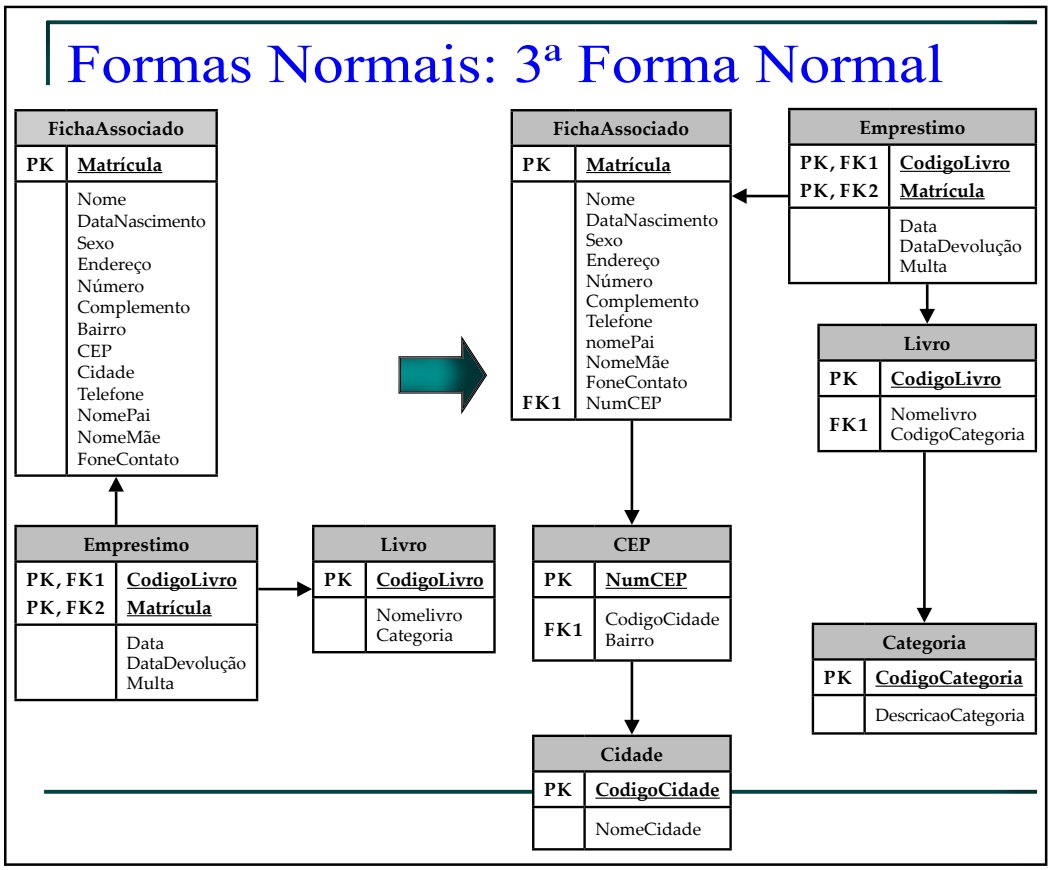
A terceira forma normal (ou 3FN) requer não haver dependências funcionais não triviais de atributos que não sejam chave em qualquer coisa, exceto um superconjunto de uma chave candidata (BATISTA, 2015).

Está na 3FN a tabela que **está na 2FN** e não contém dependências transitivas (ou indiretas):

- Dependência Transitiva: coluna não chave depende funcionalmente de outra coluna ou combinação de colunas não chave.

Evita:
Inconsistência devido à duplicidade de informações
Perda de dados em operações de remoções/alteração na relação

FIGURA 24 - TABELA FUNCIONÁRIO SEM NORMALIZAÇÃO



FONTE: A autora

Pode-se concluir, então, que ao normalizar as situações obtém-se um número maior de tabelas, porém sem problemas de redundância e inconsistência dos dados.

ENTÃO: NORMALIZAR OU NÃO NORMALIZAR?

➔ ***A decisão deve ser tomada considerando-se o compromisso entre garantir a eliminação de inconsistências na base e eficiência de acesso.***

RESUMO DO TÓPICO 3

Neste tópico, você aprendeu que:

- Modelar dados é adaptar um processo de um ambiente para o outro. Neste caso, dados. Podemos considerar como a menor parte da informação. É através de sua junção ou de um determinado contexto que o dado se transforma em informação. Então, na modelagem de dados, adaptamos os dados de um ambiente "físico" para o "computacional".

- Para efetuar a modelagem são necessários três passos (modelos): conceitual, lógico e físico.

- No modelo conceitual são levantados os requisitos dos usuários, o que ele deseja para o seu sistema. Nesta etapa não é utilizado nenhum conceito técnico para facilitar a comunicação entre os usuários e os desenvolvedores.

- Depois de definidos os requisitos dos usuários, são elaborados o conceito lógico, ou seja, a estrutura, a representação gráfica dos requisitos, com apenas o que é realmente essencial, definindo as tabelas, os campos, os tipos de valores etc.

- Não podemos esquecer dos conceitos essencias para a modelagem de dados: entidade e atributo. Entidade é uma representação do mundo físico. Pode ser abstrata ou concreta. Abstrata: estoque. Concreta: cliente.

- Atributos são as características da entidade, como a entidade cliente pode ter os seguintes atributos: nome, CPF e RG.

- Após identificar as entidades e os atributos, devemos criar o relacionamento e a cardinalidade.

- O relacionamento é a associação ou ligação entre entidades. Pode ser unária (a entidade se relaciona com ela mesma), binária (relacionamento entre duas entidades), ternária (relacionamento entre três entidades), e assim por diante.

- A cardinalidade é a quantidade de ocorrências entre as entidades. E podem ser de um para um, um para muitos e muitos para muitos.

- É através da normalização que se minimiza e a inconsistência e a redundância de dados.

- Existem diversas formas de normalização, mas as mais comuns são três, as quais são definidas da seguinte forma: Primeira Forma Normal, Segunda Forma Normal e Terceira Forma Normal.

- Na Primeira Forma Normal todos os campos devem ser atômicos, ou seja, não devem ter repetição de valores no mesmo campo.

- Na Segunda Forma Normal todos os campos devem ser dependentes da chave primária. Portanto, se algum campo não for dependente, deve-se criar uma nova tabela.

- Na Terceira Forma Normal, além de estar na Segunda Forma Normal, é preciso eliminar os campos que podem ser obtidos através de outros campos, como idade e data de nascimento.

- Após a normalização, as estruturas dos dados estão consistentes e sem redundância e, assim, eliminam problemas de atualização, manipulação e manutenção do sistema.

AUTOATIVIDADE

1) Diferencie Banco de Dados, Sistemas de Banco de Dados e Sistemas Gerenciadores de Banco de Dados.

2) Relacione três exemplos de Sistemas Gerenciadores de Banco de Dados utilizados atualmente.

3) Quais são as principais vantagens da utilização de um Sistema de Banco de Dados em relação aos sistemas tradicionais de gerenciamento por arquivo?

4) Indique pelo menos dois problemas que dificultam a utilização de SGBD.

5) Do que é composto um Banco de Dados relacional?

6) O que significa dizer que os dados de um BD estão íntegros?

7) Explique o seu entendimento em relação ao processo de modelagem de dados em Banco de Dados.

8) Desenvolva o Diagrama Entidade-Relacionamento para as seguintes situações:

Passo 1 – IDENTIFICAR AS ENTIDADES.

Passo 2 – IDENTIFICAR O RELACIONAMENTO.

1. Um aluno realiza vários trabalhos.
   Um trabalho é realizado por um ou mais alunos.

2. Um diretor dirige no máximo um departamento.
   Um departamento tem no máximo um diretor.

3. Um autor escreve vários livros.
   Um livro pode ser escrito por vários autores.

4. Uma equipe é composta por vários jogadores.
   Um jogador joga apenas em uma equipe.

5. Um cliente realiza várias encomendas.
   Uma encomenda diz respeito apenas a um cliente.

9) O que é normalização?

10) Quando uma tabela está nas 1FN, 2FN, 3FN?

11) Crie uma base de dados cujas tabelas estejam pelo menos na 3FN, apresentando os diagramas de pendência de cada tabela.

12) Considere o Banco de Dados de uma livraria. De acordo com os requisitos a seguir, utilize o MER para representar o Banco de Dados desta livraria.

a. A livraria deseja manter um cadastro de clientes.

b. Sobre cada cliente é importante manter seu endereço, telefone, CPF e lista dos livros que este cliente já comprou. Para cada compra, é importante guardar a data em que esta foi realizada.

c. Um cliente pode comprar muitos livros. Um livro pode ser vendido para mais de um cliente, pois, geralmente, há vários livros em estoque.

d. Um cliente pode ser pessoa física ou jurídica. Se for pessoa jurídica, o seu identificador deve ser o CNPJ.

e. A livraria compra livros de editoras.

f. Sobre as editoras, a livraria precisa de seu código, endereço, telefone de contato e o nome de seu gerente.

g. Cada cliente tem um código único.

h. Deve-se manter um cadastro sobre cada livro na livraria. Para cada livro, é importante armazenar o nome do autor, assunto, editora, ISBN e a quantidade dos livros em estoque.

i. Editoras diferentes não fornecem o mesmo tipo de livro.

13) Considere o Banco de Dados de um hospital. De acordo com os requisitos a seguir, utilize o MER para representar o banco de dados deste hospital.

a. O hospital possui várias alas.

b. Cada ala possui uma enfermeira responsável.

c. Cada enfermeira se reporta a uma enfermeira-chefe.

d. Enfermeiras podem atender apenas uma ala.

e. O hospital atende (credencia) os planos de saúde A, B e C.

f. Para cada plano de saúde é necessário saber os médicos credenciados nele.

g. Médico tem CRM e enfermeira tem CRE, que lhes são únicos.

h. Todo atendimento de um médico a um paciente deve ser registrado com a data e hora em que ocorreu.

h. Um mesmo paciente pode ser atendido por mais de um médico.

i. Hospital tem CNPJ.

j. Ala do hospital tem um identificador.

k. Plano de saúde tem um nome e telefone da operadora.

l. Médicos têm nome e especialidade.

m. Enfermeiras têm nome.

n. O nome de um plano de saúde é único.

UNIDADE 2

PROGRAMANDO EM BD – SQLPLUS

OBJETIVOS DE APRENDIZAGEM

A partir desta unidade você será capaz de:

- construir/entender comandos SQL de definição de dados (criação, remoção e alteração da estrutura de uma base de dados);
- construir/entender comandos SQL de manipulação de dados (inserção/remoção/alteração de dados e consultas SQL);
- compreender código SQL- DDL por trás de um modelo ER;
- construir algoritmos em Banco de Dados.

PLANO DE ESTUDOS

Esta unidade de ensino está dividida em dois tópicos, sendo que no final de cada um deles você encontrará atividades que contribuirão para a apropriação dos conteúdos.

TÓPICO 1 – SQLPLUS

TÓPICO 2 – PLSQL

TÓPICO 1

SQLPLUS

1 INTRODUÇÃO

A linguagem SQL (*Structured Query Language*) é a base para utilização de Bancos de Dados relacionais. Com a utilização dos comandos básicos (INSERT, DELETE, UPDATE e SELECT) pode-se resolver a maior parte dos problemas relacionados à manutenção e extração de dados no Banco de Dados. Com o SQL é possível criar as estruturas básicas de armazenamento, como tabelas e índices. Também há comandos específicos da linguagem para o controle e a segurança relacionados a um Banco de Dados. Em princípio, os comandos SQL são divididos em:

- DDL (*Data Definition Language*) ou Linguagem de Definição de Dados.
- DML (*Data Manipulation Language*) ou Linguagem de Manipulação de Dados.
- DQL (*Data Query Language*) ou Linguagem de Recuperação de Dados.
- DCL (*Data Control Language*) ou Linguagem de Controle de Dados.

O SQL tem sido aprimorado ao longo do tempo. Duas entidades (ANSI – *American National Standards Institute* e ISO – *International Standards Organization*) vêm, desde 1986, publicando padrões de especificação da linguagem SQL.

Vamos discutir como abordar e tirar o máximo de proveito desta linguagem que é extremamente importante para todos os profissionais de Banco de Dados. Os principais tópicos que serão de interesse envolvem:

- Fundamentos da linguagem SQL.
- Melhoria no desempenho de consultas.
- Aplicação de funções e comandos em buscas.

2 PROGRAMAÇÃO DE BANCO DE DADOS

Os comandos da linguagem SQL são muito poderosos, mas normalmente consegue-se melhorar o desempenho das aplicações através da programação do Banco de Dados. Ao desenvolver módulos que sejam executados diretamente no servidor, diminui-se o tráfego de informações na rede, esconde-se boa parte das estruturas das tabelas e agiliza-se o processamento e retorno das mensagens. Internamente, o Banco de Dados possui mecanismos integrados que permitem unir as estruturas tradicionais de programação com os comandos SQL.

Leia mais em: SQL e Programação de Banco de Dados. Disponível em: <http://www.devmedia.com.br/sql-e-programacao-de-banco-de-dados/3139#ixzz3RGT16lxk>

3 CONCEITOS BÁSICOS

As seguintes definições explicam conceitos fundamentais:

QUADRO 1 - RESUMO DOS CONCEITOS DE BANCO DE DADOS

Comando (Command)	Uma instrução dada ao SQL*Plus ou ORACLE.
(Table)	Unidade básica de armazenamento no ORACLE.
Linha (Row)	Uma linha é uma combinação dos valores da coluna em uma tabela. Às vezes, linha é chamada de registro.
Coluna (Column)	Uma coluna corresponde a um tipo de dado em uma tabela. É descrita por um nome e mantém os dados de um tipo e tamanho específicos.
Campo	Na interseção de uma linha com uma coluna há um campo.
Chave primária	É a coluna ou o conjunto de colunas que identifica com exclusividade cada linha em uma tabela.
Chave estrangeira	Uma chave estrangeira é uma coluna ou conjunto de colunas referente a uma chave primária na mesma ou em outra tabela.
Bloco (Block)	Um grupo de comandos SQL e PL/SQL dentro de uma lógica procedural.
Consulta (Query)	Comando SQL (Select) que recupera informação de uma ou mais tabelas.

FONTE: A autora

Regras Gerais:

- Valores duplicados não são aceitos em uma chave primária.

Lista resumida dos principais objetos do Banco de Dados:

QUADRO 2 - PRINCIPAIS OBJETOS DO BD

Objeto	Descrição
Table	Unidade básica de armazenamento composta de linhas e colunas.
View	Representa, logicamente, subconjuntos de dados de uma ou mais tabelas.
Sequence	Gera valores de chave primária.
Index	Melhora o desempenho de algumas consultas.
Synonym	Nome alternativo para um objeto.
Program unit	Procedimento, função, ou pacote de comando SQL e PL/SQL.

FONTE: A autora

Os comandos SQL, SQL*Plus e PL/SQL são usados para acessar e tratar dados armazenados em um Banco de Dados Oracle.

QUADRO 3 - DIFERENÇAS ENTRE FERRAMENTAS E LINGUAGENS

Linguagem ou ferramenta	Descrição
SQL	Uma linguagem de comandos para comunicação com o Oracle a partir de qualquer ferramenta ou aplicação.
SQL*Plus	Uma ferramenta Oracle que reconhece, submete seus comandos SQL e PL/SQL para execução no Server.
PL/SQL	Uma linguagem procedural Oracle para escrita da lógica da aplicação e manipulação de dados fora do banco de dados.

FONTE: A autora

EFETUANDO O LOG-IN NO SQL*PLUS

O modo de inicialização do SQL*Plus depende do tipo de sistema operacional ou do ambiente Windows em uso.

No Windows:

FIGURA 25 – LOG-IN NO BANCO DE DADOS

FONTE: A autora

Onde:

Username: é o nome do usuário no banco.
Password: é a senha do usuário no banco.
Host String: é a *string* de conexão ao banco.

Via linha de comando:

```
SQLPLUS [USERNAME [/PASSWORD [@DATABASE]]]
Ex:
/>sqlplus scott/tiger@bd01p
```

EXECUTANDO COMANDOS NO SQL*PLUS

No *prompt* de comando (SQL>), digite a primeira linha de comando:

➔ Exemplo:

```
SQL>   create table solic_compra
     (num_solic number(10) not null
     ,dat_emiss date
     ,dat_previsao date
     ,deptno number(02));
```

Termine o comando com 'ponto e vírgula' ou tecle <enter>, abrindo nova linha e digite / e enter.

O SQL*Plus armazena o conteúdo do último comando dentro do buffer SQL.

4 COMANDOS DE MANIPULAÇÃO DO CONTEÚDO DO BUFFER SQL

QUADRO 4 - COMANDOS PARA USO NO BUFFER DO SQL*PLUS

Comando	Sintaxe
List	L[ist] [n] [n m] [n *] [n last] [* n] [* last] [last] [*]
Input	I[nput] [texto]
Change	C[hange] / texto_antigo / texto_novo/
Del	Del [n] [n m] [n *] [n last] [* n] [* last] [last]
Append	A[ppend] texto
Spool	SPOOL nome_de_arquivo – inicia geração do arquivo especificado SPOOL OFF – encerra geração do arquivo
Edit	ED[it] [nome_de_arquivo]
Get	GET nome_de_arquivo [list] [nolist]
Start ou @	START file_name[.ext] [arg1 arg2 ...] @ file_name[.ext] [arg1 arg2 ...]

FONTE: A autora

5 MANIPULAÇÃO DE OBJETOS NO BANCO DE DADOS

EXIBINDO A ESTRUTURA DA TABELA

Através do SQL*Plus é possível exibir a estrutura de uma tabela usando o comando DESCRIBE. Serão apresentados os nomes das colunas, tipos de dados e se a coluna pode receber valores nulos.

Sintaxe: desc[ribe] tablename

Exemplo:

```
SQL> desc dept
 Name                  Null?    Type
 ----------------------- -------- ----------------------------
DEPTNO                   NOT NULL NUMBER(2)
 DNAME                            VARCHAR2(14)
 LOC                              VARCHAR2(13)
```

6 SELECIONANDO E RECUPERANDO DADOS DE UMA TABELA

Através do SQL*Plus é possível recuperar colunas de uma ou mais tabelas, selecionar linhas que obedeçam a determinados critérios, realizar operações em relação às colunas, enfim, obter as informações desejadas provenientes das tabelas.

Para tal, utiliza-se o comando SELECT, que é o comando mais utilizado, sendo um dos mais importantes da linguagem SQL.

Sintaxe: SELECT [ALL / DISTINCT]
{*[nome_coluna[,função(nome_coluna)][,'texto']}
FROM <nome_tabela>
WHERE <critério de seleção>
GROUP BY
HAVING
ORDER BY <coluna1>,<coluna2>,<colunaN>

DISTINCT – Não mostra várias linhas com os mesmos valores de resultado, somente uma linha é mostrada.

FROM – Tabelas onde os dados são buscados.

WHERE – Condições para seleção dos dados.

GROUP BY – Colunas pelas quais os dados devem ser agrupados. Somente é utilizado em conjunto com as funções de grupo.

HAVING – Condições para seleção dos dados utilizando as funções de grupo.

ORDER BY – Colunas que definem a ordem de classificação das linhas de resultado.

➔ Exemplo:

```
SQL> select * from emp
        where deptno = 10
        order by empno;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7782 CLARK      MANAGER         7839 09-JUN-81       2450          10
      7839 KING       PRESIDENT            17-NOV-81       5000          10
      7934 MILLER     CLERK           7782 23-JAN-82       1300          10

3 rows selected.
```

Observações:

- O caractere '*' (asterisco) indica que serão exibidas todas as colunas.

- A cláusula WHERE pode trabalhar com vários tipos de comparação, permitindo combinações para uma restrição mais específica para a escolha das linhas.

7 AS VISÕES DO DICIONÁRIO DE DADOS

As informações referentes à estrutura do próprio Banco de Dados também estão armazenadas em tabelas. O conjunto dessas tabelas é chamado de dicionário de dados, e as principais informações que podemos extrair são:

Tabela ALL_OBJECTS
Contém informações sobre os objetos armazenados no Banco de Dados.
Principais colunas: OBJECT_NAME, OBJECT_TYPE, OWNER

Tabela ALL_TABLES
Contém informações sobre as tabelas do Banco de Dados.
Principais colunas: TABLE_NAME, OWNER

Tabela ALL_TAB_COLUMNS
Contém informações sobre as colunas de cada tabela.
Principais colunas: TABLE_NAME, COLUMN_NAME, DATA_TYPE, DATA_LENGTH, NULLABLE

Tabela ALL_INDEXES
Contém informações sobre os índices das tabelas.
Principais colunas: INDEX_NAME, TABLE_NAME, STATUS

Tabela ALL_IND_COLUMNS
Contém informações sobre as colunas que compõem os índices.
Principais colunas: TABLE_NAME, INDEX_NAME, COLUMN_NAME, POSITION

Tabela ALL_CONSTRAINTS
Contém informações sobre as constraints das tabelas.
Principais colunas: OWNER, CONSTRAINT_NAME, CONSTRAINT_TYPE, R_CONSTRAINT_NAME, STATUS

Tabela ALL_CONS_COLUMNS
Contém informações sobre as colunas que compõem as constraints.
Principais colunas: CONSTRAINT_NAME, TABLE_NAME, COLUMN_NAME, POSITION

Tabela ALL_TRIGGERS
Contém informações sobre as triggers das tabelas.
Principais colunas: TRIGGER_NAME, TRIGGER_TYPE, TABLE_NAME, WHEN_CLAUSE, STATUS, TRIGGER_BODY

Tabela ALL_TRIGGER_COLS
Contém informações sobre as colunas que compõem as triggers.
Principais colunas: TABLE_NAME, TRIGGER_NAME, COLUMN_NAME.

Tabela ALL_SOURCE
Contém informações sobre código PL/SQL armazenado na base de dados.
Principais colunas: NAME, TYPE, LINE, TEXT

Tabela ALL_USERS
Contém informações sobre os usuários existentes na base de dados.

OPERADORES SQL*PLUS

Igualdade: =
Ex.: select * from emp where deptno = 10;

Diferença: != ou ^= ou <>
Ex.: select * from emp where deptno < > 10;

Maior que: >
Ex.: select * from emp where deptno > 10;

Maior ou igual: >=
Ex.: select * from emp where deptno > = 10;

Menor que: <
Ex.: select * from emp where deptno < 12;

Menor ou igual: <=
Ex.: select * from emp where deptno < = 12;

Nulo: IS NULL
Ex.: select * from emp where mgr is null;

Não é nulo: IS NOT NULL
Ex.: select * from emp where mgr is not null;

Pertence ao conjunto: IN
Ex.: select * from emp where empno in (7369,7499,7521,7566);

Não Pertence ao conjunto: NOT IN
Ex.: select * from emp
where empno not in (7369,7499,7521,7566);

Pertence a um intervalo: BETWEEN
Ex.: select * from emp where empno between 7369 and 7566;

Não pertence a um intervalo: NOT BETWEEN
Ex.: select * from emp
where empno not between 7369 and 7566;

Semelhante: LIKE
Podem ser utilizados caracteres coringas:
% - Qualquer sequência de caracteres (zero, um ou mais caracteres)
_ - Qualquer caractere (somente um caractere)
Ex.: select * from emp where job like '%SALE%';

Não semelhante: NOT LIKE'
Ex.: select * from emp where job not like '%SALE%';

Existe: EXISTS
Somente é utilizado em conjunto com subquery
Ex.: select a.deptno, a.dname from dept a
where exists (select 1 from solic_compra b
where b.deptno = a.deptno);

Não existe: NOT EXISTS
Somente é utilizado em conjunto com subquery
Ex.: select a.deptno, a.dname from dept a
where not exists (select 1 from solic_compra b
where b.cod_depto = a.deptno);

Operadores Lógicos: AND, OR, NOT
Ex.: select * from emp
where ((job = 'CLERK' and sal > 1000)
or (job = 'ANALYST' and sal > 2000));

8 PERSONALIZANDO O AMBIENTE SQL*PLUS ATRAVÉS DE VARIÁVEIS DO SISTEMA

FEEDBACK

Toda vez que um comando SQL é executado, o RDBMS retorna uma mensagem de acordo com o tipo de operação que foi realizada. Essas mensagens podem ser ou não omitidas através deste comando (continua mostrando mensagem de erro). Sintaxe: Set Feed[back] on/off/

SPACE

Controla a quantidade de caracteres em branco que devem ser exibidos entre uma coluna e outra. Sintaxe: Set Spa[ce] número de colunas.

LINESIZE

Define o número máximo de caracteres que será exibido por linha. Sintaxe: Set Lin[esize] número de caracteres(set lines 100).

WRAP

Se o total de caracteres a serem exibidos for maior que o número máximo especificado na variável de sistema LINESIZE, o SQL*Plus passará à(s) coluna(s) que não couber na linha para a linha de baixo (WRAP = ON), do contrário o restante da linha será truncado (WRAP = OFF). Sintaxe: Set wra[p] on/off (se continua embaixo ou trunca).

HEADING

Toda a coluna selecionada carrega um título de identificação. Caso seja necessária a omissão dele, basta atribuir OFF à variável de sistema HEADING. Sintaxe: Set hea[ding] on/off.

UNDERLINE

Define o caractere que sublinha o título de identificação, o qual não pode ser alfanumérico nem espaço em branco. Sintaxe: Set und[erline] caracter. Set underline OFF coloca Título e campos juntos.

PAGES[IZE]

Define o tamanho da página. Sintaxe: Set pages[ize] número de linhas (set pages 60).

PAUSE

Faz com que o SQL*Plus pare de exibir os dados selecionados toda vez que existir uma quebra de página, aguardando que o usuário pressione <enter> para exibir a próxima página.Sintaxe: Set Pau[se] on/off

9 FUNÇÕES DE MANIPULAÇÃO

Pode-se utilizar funções das mais diversas naturezas para obter o resultado desejado em um formato desejado. Dentro das funções mais utilizadas temos:

Funções Aritméticas (de grupo):

AVG

Retorna a média dos valores da coluna, ignorando os valores nulos.
Sintaxe: AVG(coluna).

➔ Exemplo:

```
SQL> select avg(sal) from emp;

AVG(SAL)
---------
2073.2143
```

SUM

Soma os valores da coluna, ignorando os valores nulos.
Sintaxe: SUM(coluna).

➔ Exemplo:

```
SQL> select sum(sal) from emp;

SUM(SAL)
---------
   29025
```

MAX

Retorna o maior valor da coluna.
Sintaxe: MAX(coluna).

➔ Exemplo:

```
        SQL> select max(sal) from emp; MAX(SAL)
---------
    5000
```

MIN

Retorna o menor valor de uma coluna.
Sintaxe: MIN(coluna).

➔ Exemplo:

```
          SQL> select min(sal) from emp;

MIN(SAL)
---------
     800
```

COUNT

Conta quantos elementos não nulos existem em uma coluna.
Sintaxe: COUNT(coluna).

➔ Exemplo:

```
          SQL> select count(empno) from emp;

COUNT(EMPNO)
------------
        14
```

STDDEV

Retorna o desvio padrão de uma coluna.
Sintaxe: STDDEV(coluna).

➔ Exemplo:

```
          SQL> select stddev(sal) from emp;

STDDEV(SAL)
-----------
  1182.5032
```

10 FUNÇÕES DE FORMATO

TO_CHAR

Converte um valor do tipo data para um formato alternativo especificado.
Sintaxe: TO_CHAR (Date, 'Formato')

➔ Exemplo:

```
SQL> select num_solic, to_char(dat_emiss, 'dd/mm/yyyy')
    from solic_compra;
```

Abaixo são mostrados os principais formatos que podem ser utilizados:

QUADRO 5 - PRINCIPAIS FORMATOS PARA CAMPOS DO TIPO DATA

Formato	Significado
CC	Século
Y	Último dígito do ano
YY	Últimos 2 dígitos do ano
YYY	Últimos 3 dígitos do ano
YYYY	Ano com 4 dígitos
Q	Trimestre da data
MONTH	Mês por extenso (em inglês)
MON	Três primeiras letras do mês por extenso (em inglês)
MM	Número do mês
RM	Número do mês (em algarismos romanos)
WW	Número de semanas desde o 1º dia do ano
W	Número de semanas desde o 1º dia do mês
DD	Dia do mês
DDD	Dia do ano (sistema juliano de datas)
D	Dia da semana (1 = Domingo; 2 = Segunda; 3 = Terça)
DAY	Dia da semana por extenso (em inglês)
DY	Três primeiras letras do dia da semana por extenso (em inglês)
HH ou HH12	Hora no sistema de 12 horas
HH24	Hora no sistema de 24 horas
PM ou P.M.	Mostra a convenção AM (até meio-dia) ou PM (após meio-dia)
MI	Minutos
SS	Segundos
SSSSS	Segundos desde o início do dia

FONTE: A autora

Os caracteres - / , . ; : "texto" são reproduzidos no resultado.
Também é utilizado para converter o tipo de dado Number para tipo Char.

Sintaxe: TO_CHAR (Number, 'Formato')
Prefixo: B → valor zerado exibido em branco
Sufixo: MI → se valor(-), sinal no final do número
PR se valor(-), formato <n>.

➔ Exemplo:

```
SQL> select ename, to_char(sal,'999,999.99')
     from emp where empno = 7499;

ENAME      TO_CHAR(SAL
---------- -----------
ALLEN        1,600.00
```

A seguir são mostrados os principais formatos que podem ser utilizados:

QUADRO 6 - PRINCIPAIS FORMATOS PARA NÚMEROS

Formato	Significado
9	Mostra o número com a quantidade de dígitos especificada. Zeros à esquerda são mostrados como caractere em branco. Caso o número for zero, o último caractere é mostrado como caractere '0'. Números negativos incluem o sinal antes do número.
0	Mostra zeros à esquerda como caractere '0'.
$	Mostra o número precedido do caractere '$'.
B	Zeros à esquerda são mostrados como caractere em branco. Caso o número for zero, o último caractere também é mostrado como caractere em branco.
MI	Mostra números negativos com o sinal de negativo após o número.
S	Mostra o sinal do número, seja positivo ou negativo.
PR	Mostra número negativo envolvido pelos caracteres '<' e '>'.
D ou .	Mostra o ponto decimal.
G ou ,	Mostra o separador de milhar.
RN	Mostra o número em algarismos romanos.
FM	Mostra número sem caracteres em branco, antes ou após o número.

FONTE: A autora

TO_DATE

Permite formatar para outros formatos de datas.
Sintaxe: TO_DATE (Date,'Date Picture')

➔ Exemplo:

```
SQL> select to_char(to_date('01011997','DDMMYYYY'),'Month DD, YYYY')
from dual;

TO_CHAR(TO_DATE('0
------------------
January   01, 1997
```

TO_NUMBER

Converte uma string em um tipo Number.
Sintaxe: TO_NUMBER (‘string’).

➔ Exemplo:

```
SQL> select * from emp
    where sal > to_number('2000');

 EMPNO ENAME     JOB         MGR HIREDATE      SAL     COMM   DEPTNO
--------- ---------- --------- --------- --------- --------- --------- ---------
    7839 KING       PRESIDENT           17-NOV-81      5000                10
```

CONCATENAÇÃO (||)

Utilizado para concatenar duas colunas.
Sintaxe: coluna || ‘string’ ||

➔ Exemplo:

```
SQL> select deptno || '--->' || dname from dept;

DEPTNO||'--->'||DNAME
----------------------
10--->ACCOUNTING
20--->RESEARCH
30--->SALES
40--->OPERATIONS
```

Funções para valores alfanuméricos

SUBSTR

Retorna uma parte de um char inicimando em *m*, *n* caracteres.
Sintaxe: Substr(char,m,n)

➔ Exemplo:

```
SQL> select substr('Oracle SQL',1,6) from dual;

SUBSTR
------
Oracle
```

INSTR

Procura em *Char1* iniciando na posição *n,* a *m*-ésima ocorrência de *char2*.
Caso n for negativo, procura de trás para frente.
Sintaxe: Instr(char1,char2[,n[,m]])

➔Exemplo:

```
SQL> select instr('Blumenau','na',1,1) from dual;


INSTR('BLUMENAU','NA',1,1)
--------------------------
```

REPLACE

Substitui *char*1 buscando por *char2* substituindo por *char3*.
Sintaxe: Replace(char1,char2,[char3])

➔ Exemplo:

```
SQL> select replace ('Comparação', 'mpa', 'ope')
from dual;

REPLACE('CO
-----------
Cooperação
```

LTRIM

Remove o conteúdo de *set* da esquerda de *char1*. Por default, *set* é o caractere de espaço em branco.
Sintaxe: Ltrim(char1,[set])

➔ Exemplo:

```
SQL> select ltrim('Informática','Info') from dual;

LTRIM('
-------
rmática

SQL> select ltrim('   Informática') from dual;

LTRIM('INF
----------
Informática
```

RTRIM

Remove o conteúdo de *set* da direita de *char1*. Por default, *set* é o caracter de espaço em branco.
Sintaxe: Rtrim(char1,[set])

➔ Exemplo:

```
SQL> select rtrim('informática','ica') from dual;

RTRIM('I
--------
informát

SQL> select rtrim('Informática   ') from dual;

RTRIM('INF
-----------
Informática
```

LENGTH

Retorna o número de caracteres contido em *char*.
Sintaxe: Length(char)

➔ Exemplo:

```
SQL> select length('Informática') from dual;

LENGTH('INFORMÁTICA')
---------------------
11
```

UPPER

Retorna *char* com todos os caracteres em formato maiúsculo.
Sintaxe: Upper(char)

➔ Exemplo:

```
SQL> select upper('blumenau') from dual;

UPPER('B
--------
BLUMENAU
```

11 FUNÇÕES PARA VALORES NUMÉRICOS

ABS

Retorna o valor absoluto de *n*.
Sintaxe: ABS(n)

➔ Exemplo:

```
SQL> select abs(-32.98) from dual;

ABS(-32.98)
-----------
   32.98
```

MOD

Retorna o resto da divisão de *m* por *n*.
Sintaxe: Mod(m,n)

➔ Exemplo:

```
SQL> select mod(11,3) from dual;

MOD(11,3)
--------------
         2
```

ROUND

Retorna *n* arredondado para *m* casas, segundo o arredondamento universal. Números abaixo de 5 são arredondados para baixo. Números de 5 para cima são arredondados para cima. Caso *m* for omitido, arredonda para inteiro. Caso *m* for negativo, arredonda as casas da parte inteira do número.
Sintaxe: Round(n[,m])

➔ Exemplo:

```
SQL> select round(3.566233,2) from dual;

ROUND(3.566233,2)
-----------------
  3.57

SQL> select round(153.566233,-2) from dual;

ROUND(153.566233,2)
-------------------
   200
```

SIGN

Se *n* < 0, a função retorna a -1; se *n* = 0, a função retorna a 0; e se *n* > 0, a função retorna a 1.
Sintaxe: Sign(n)

➔ Exemplo:

```
SQL> select sign(-23) from dual;

SIGN(-23)
---------
-1
```

TRUNC

Retorna ao valor de *n* truncado para *m* casas decimais, ou seja, sempre arredondando para baixo. Caso *m* for omitido, o valor inteiro de *n* é retornado. Caso *m* for negativo, arredonda as casas da parte inteira do número Sintaxe: TRUNC(n[,m])

➔ Exemplo:

```
SQL> select trunc(3.566233,2) from dual;

TRUNC(3.566233,2)
-----------------
             3.56

SQL> select trunc(153.566233,-2) from dual;

TRUNC(153.566233,2)
-------------------
   100
```

12 FUNÇÕES PARA DATAS

ADD_MONTHS

Retorna à data *d* mais *n* meses.
Sintaxe: ADD_MONTH(d,n)

➔ Exemplo:

```
SQL> select add_months(sysdate,3) from dual;

   ADD_MONTH
----------
04-MAY-97
```

LAST_DAY

Retorna à data do último dia do mês em relação à data *d.*
Sintaxe: Last_day(d)

➔ Exemplo:

```
SQL> select sysdate, last_day(sysdate) from dual;

SYSDATE     LAST_DAY(
----------- ----------
04-FEB-97   28-FEB-97
```

MONTHS_BETWEEN

Retorna ao número de meses entre *d2* e *d1*. Caso *d2* seja maior que *d1*, o valor retornado será positivo.
Sintaxe: Months_Between(d1,d2)

➔ Exemplo:

```
SQL> select months_between(to_date('01012000','MMDDYYYY'),
 2  to_date('02041997','MMDDYYYY'))
 3  "Meses Restantes" from dual;

Meses Restantes
---------------
     34.90323
```

SYSDATE

Retorna à data e hora corrente.
Sintaxe: **sysdate**

➔ Exemplo:

```
SQL> select sysdate from dual;

SYSDATE
---------
04-FEB-97
```

13 FUNÇÕES GENÉRICAS

GREATEST

Retorna ao maior da lista de expressões.
Sintaxe: Greatest(exp[,exp2...])

➔ Exemplo:

```
SQL> select greatest(4,8,3) from dual;

GREATEST(4,8,3)
---------------
              8
```

LEAST

Retorna ao menor valor da lista de expressões.
Sintaxe: Least(exp[,exp2...])

➔ Exemplo:

```
SQL> select least(4,8,3) from dual;

LEAST(4,8,3)
------------
           3
```

DECODE

Retorna a um valor decodificado de acordo com uma comparação.
Sintaxe: Decode(valor, condição 1, resposta1, condição 2, resposta2, resposta 3)
Nesse caso, resposta 3 é atribuída caso nenhuma das comparações seja verdadeira.

➔ Exemplo:

```
SQL> select empno, decode(comm,null,'SEM COMISSAO', 'OK')
  2  from emp
  3  where deptno = 30;

   EMPNO DECODE(COMM,
--------- ------------
    7499 OK
    7654 OK
    7698 SEM COMISSAO
    7844 OK
    7900 SEM COMISSAO
```

NVL

Retorna a um valor específico se o valor de retorno for nulo.
Sintaxe: Nvl(coluna, valor)

➔ Exemplo:

```
SQL> select empno, nvl(comm,0) from emp where deptno = 30;

    EMPNO NVL(COMM,0)
--------- -----------
     7499         300
     7654        1400
     7698           0
     7900           0
```

14 CRIANDO E ALTERANDO A ESTRUTURA DE UMA TABELA

CREATE TABLE: Comando de criação de tabela, na qual o usuário informa que colunas (nome, tipo e tamanho) pertencem à tabela, e também que restrições deverão ser tratadas para cada coluna ou conjunto de colunas.

Nome da tabela: no máximo 30 caracteres.
Nome da coluna: no máximo 30 caracteres.

Sintaxe:

```
CREATE   TABLE nome_tabela
(<nome_coluna> <tipo_dado (tamanho)> <restrições>,
<nome_coluna> <tipo_dado (tamanho)> <restrições>,
...);
```

➔ Exemplo:

```
SQL> create table produto
 2  (cod_produto   number(07)   not null primary key,
 3   dsc_produto   varchar2(30) not null,
 4   dsc_unidade   varchar2(10) not null);

Table created.
```

OBS.: Para verificar as colunas da tabela: DESC tabela[.coluna]

Tipos de dados

QUADRO 7 - PRINCIPAIS TIPOS DE DADOS DA BASE DE DADOS

Datatype	Descrição
CHAR	Um campo de caractere de comprimento fixo com até 2000 *bytes* de comprimento.
NCHAR	Um campo de comprimento fixo para conjuntos de caracteres *multibyte*. O tamanho máximo é de 2000 caracteres ou 2000 *bytes*, dependendo do conjunto de caracteres.
VARCHAR2	Um campo de caractere de tamanho variável, com até 4000 caracteres de comprimento.
NVARCHAR2	Um campo de comprimento variável para conjuntos de caracteres *multibytes*. O tamanho máximo é de 4000 caracteres ou 4000 *bytes*, dependendo do conjunto de caracteres.
DATE	Um campo de comprimento fixo de 7 *bytes* usados para armazenar todas as datas. O tempo é armazenado como parte da data. Quando consultada, a data estará no formato DD-Mon-YY, como em 15-OCT-99 para 15 de outubro de 1999, a menos que o parâmetro NLS_DATE_FORMAT seja definido no INIT. ORA.
NUMBER	Uma coluna de número de comprimento variável. Os valores permitidos são zero e números positivos e negativos. Os valores NUMBER geralmente são armazenados em 4 ou menos *bytes*.
LONG	Um campo de comprimento variável com até 2GB de comprimento.
RAW	Um campo de comprimento variável usado para dados binários de até 2000 *bytes* de comprimento.
LONG RAW	Um campo de comprimento variável usado para os dados binários com até 2GB de comprimento.
BLOB	*Binary large object*, com até 4 GB de comprimento.
CLOB	*Character large object*, com até 4 GB de comprimento.
NCLOB	Datatype CLOB para conjuntos de caracteres *multibyte* e até 4GB de comprimento.
BFILE	Arquivo externo binário, o tamanho é limitado pelo sistema operacional.
ROWID	Dados binários que representam o RowID. O valor será de 10 *bytes*.

FONTE: A autora

Observação:

Os campos Long e Long Raw apresentam as seguintes restrições:

1 - Não podem fazer parte de cláusulas where, group by, connect by ou distinct, nem em expressões (Substr, Instr etc.).
2 - Não podem ser usadas como variáveis ou argumentos em um bloco PL.

ALTERANDO A ESTRUTURA DE UMA TABELA

Após a criação de uma tabela podemos acrescentar colunas, acrescentar ou excluir restrições e modificar as características de uma determinada coluna. Para isto, deve-se utilizar o comando ALTER TABLE.

```
Sintaxe:  ALTER TABLE nome_tabela
ADD ( nome_coluna tipo_dado [tamanho] [restrição] ,
nome_coluna tipo_dado [tamanho] [restrição] )
MODIFY (nome_coluna tipo_dado [tamanho] ,
        nome_coluna tipo_dado [tamanho] )
DROP CONSTRAINT nome_restrição
```

ADD - Acrescenta colunas ou restrições.
MODIFY - Altera definições de colunas existentes ou acrescenta a restrição de nulo ou não nulo.
DROP CONSTRAINT - Exclui restrições de colunas ou de tabelas.

➔ Exemplo:

```
SQL> alter table produto
     2 add    (vlr_unit     number(12,2) not null)
     3 modify (dsc_produto  varchar2(35));

Table altered.

SQL> alter table produto
     2  add (constraint produto_pk primary key (cod_produto));

Table altered.

SQL> alter table item_solicitacao
add (constraint item_produto_fk foreign key (cod_produto)
references produto (cod_produto));

Table altered.
```

15 INTEGRIDADE REFERENCIAL (CONSTRAINTS)

CONCEITO DE RESTRIÇÕES

São regras ou instruções informadas ao banco de dados com a finalidade de controlar o preenchimento de uma ou mais colunas de uma determinada tabela.

Estas restrições têm como função manter a integridade do Banco de Dados através da opção Constraint.

O uso de Constraints reduz consideravelmente erros de programação, pois toda integridade é controlada pelo ORACLE. Estas Constraints são armazenadas no dicionário de dados.

A violação de qualquer restrição retorna a mensagens de erros que podem ser tratadas pela aplicação.

16 TIPOS DE CONSTRAINTS

Primary Key

Define as colunas de uma determinada tabela que formam a sua chave primária. Lembre-se de que a chave primária é única, ou seja, não pode haver duas ocorrências iguais dentro da mesma tabela.

➔ Exemplo:

```
SQL> create table pedido
  2 (num_pedido   number(07) not null constraint pedido_pk primary key,
  3 cod_fornec   number(07) not null,
  4 dat_emiss          date      not null,
  5 dat_entreg  date      not null);

Table created.
```

Outra alternativa é usar a sintaxe:
ALTER TABLE tabela ADD CONSTRAINT nome PRIMARY KEY (colunas)

Unique Key

Define que coluna ou combinação de colunas não podem ter valores repetidos nas linhas de uma determinada tabela.

➔ Exemplo:

```
SQL> create table pedido
       2        (num_pedido  number(07) not null constraint pedido_pk unique,
       3                 cod_fornec  number(07) not null,
       4        dat_emiss   date      not null,
       5        dat_entreg  date      not null);

Table created.
```

Outra alternativa é usar a sintaxe:
ALTER TABLE tabela ADD CONSTRAINT nome UNIQUE (colunas)

Diferença entre Primary Key e Unique Key:

* Primary Key não permite a utilização de colunas Null.
* Unique Key permite utilização de colunas Null.
Na prática isso significa que as colunas que compõem a unique key não podem ter valor repetido, exceto o nulo que pode ser repetido em várias linhas.

References

Define que uma coluna ou um conjunto de colunas de uma tabela referencia uma primary key de outra tabela (vide foreing key). É declarada junto à coluna

➔ Exemplo:

```
SQL> create table pedido
        2 (num_pedido   number(07) not null primary key,
        3 cod_fornec   number(07) references fornecedor(cod_fornec),
        4 dat_emiss    date       not null,
        5 dat_entreg   date       not null);

Table created.
```

Foreign Key

Define que uma ou mais colunas são primary key ou unique key em outra tabela. É declarada depois de todas as colunas.

➔ Exemplo:

```
SQL> create table pedido
        2 (num_pedido  number(07) not null primary key,
        3 cod_fornec  number(07) not null,
        4 dat_emiss   date       not null,
        5 dat_entreg  date       not null,
                constraint pedido_fornec_fk foreign key (cod_fornec)
                references fornecedor(cod_fornec));
```

Table created.
Outra alternativa é usar a sintaxe:
ALTER TABLE tabela ADD CONSTRAINT nome FOREIGN KEY (colunas) REFERENCES tabela referenciada (colunas).

Check

Define um conjunto de valores permitidos para uma determinada coluna.

Esse conjunto de valores é chamado de domínio.

➔ Exemplo:

```
SQL> create table pedido
       2 (num_pedido  number(07) not null primary key,
       3 cod_fornec  number(07) not null,
       4 dat_emiss   date       not null,
       5 dat_entreg  date       not null,
       6 constraint dat_entreg_ck
       7 check  (dat_entreg between dat_emiss and  dat_emiss + 5));
```

Observação:

Not Null e Null
Define se o preenchimento da coluna é ou não é obrigatório. Caso não seja informado, é assumida a opção Null.

17 TRANSAÇÕES COM O BANCO DE DADOS

Transação é uma unidade lógica de trabalho composta por comandos de SQL que serão tratados pelo RDBMS. Uma transação começa com o primeiro comando SQL após a conexão com o SQL*PLUS ou após o encerramento da transação anterior e pode ser finalizada de diversas maneiras. Comandos que alteram o dicionário de dados (SQL DDL) não podem ser desfeitos. Estes comandos encerram uma transação, e tornam visíveis para todos os usuários as atualizações feitas no Banco de Dados pelo usuário que emitiu o comando SQL DDL (HEUSER, 2004).

Comandos que realizam operações com linhas de tabelas podem ser desfeitos com o comando ROLLBACK ou confirmados com o comando COMMIT, finalizando desta forma a transação que estava pendente. Enquanto a transação não for concluída, as alterações só serão vistas pela sessão corrente, logo, os outros usuários não terão acesso a estas modificações. Uma transação pode ser subdividida com o comando SAVEPOINT. A finalidade desta divisão é a opção de desfazer somente uma parte da transação e não toda ela.

18 SINTAXE DOS COMANDOS

SAVEPOINT nome_ponto

Por default, o número máximo de pontos ativos numa transação é 5, mas pode ser alterado para até 255.

ROLLBACK [TO [SAVEPOINT] nome_ponto **]**

Se for digitado apenas a palavra ROLLBACK, todas as transações não efetivas serão desfeitas. Caso isto não seja desejado, deve-se informar o ponto até o qual se deseja desfazer.

COMMIT

Efetiva todas as transações pendentes. Pode-se estabelecer para uma sessão, ou parte dela, que ao final da execução de cada comando de SQL DML será realizado um COMMIT AUTOMÁTICO. Desta forma cada transação será composta de um e somente um comando e não poderá ser desfeita. Para isto, deve-se atribuir ON ou IMM[EDIATE] à variável de sistema AUTO[COMMIT].

➔ Exemplo:

```
SQL> select * from solic_compra;

NUM_SOLIC COD_DEPTO DAT_EMISS  DAT_PREVIS
--------- --------- ---------- ----------
        1         1 12-01-1996 12-02-1996
        2         3 12-12-1997 13-12-1997
        3         7 12-01-1996 12-01-1997
        4         2 01-04-1997 02-04-1997
        6         3 12-01-1996 12-01-1996

SQL> insert into solic_compra
 2 (num_solic
 3 ,dat_emiss
 4 ,dat_previsao
 5 ,cod_depto)
6 values
 7 (7
 8 ,sysdate
 9 ,sysdate + 15
 10 ,1);

1 row created.

SQL> select * from solic_compra;
NUM_SOLIC COD_DEPTO DAT_EMISS  DAT_PREVIS
--------- --------- ---------- ----------
        1         1 12-01-1996 12-02-1996
        2         3 12-12-1997 13-12-1997
        3         7 12-01-1996 12-01-1997
        4         2 01-04-1997 02-04-1997
```

```
         6          3 12-01-1996 12-01-1996
         7          1 03-08-2001 18-08-2001

6 rows selected.

SQL> rollback;

Rollback complete.

SQL> select * from solic_compra;

NUM_SOLIC COD_DEPTO DAT_EMISS  DAT_PREVIS
--------- --------- ---------- ----------
         1          1 12-01-1996 12-02-1996
         2          3 12-12-1997 13-12-1997
         3          7 12-01-1996 12-01-1997
         4          2 01-04-1997 02-04-1997
         6          3 12-01-1996 12-01-1996

SQL> insert into solic_compra
 2 (num_solic
 3 ,dat_emiss
 4 ,dat_previsao
 5 ,cod_depto)
 6 values
 7 (8
 8 ,sysdate + 1
 9 ,sysdate + 12
 10 ,3);

1 row created.

SQL> select * from solic_compra;
NUM_SOLIC COD_DEPTO DAT_EMISS  DAT_PREVIS
--------- --------- ---------- ----------
         1          1 12-01-1996 12-02-1996
         2          3 12-12-1997 13-12-1997
         3          7 12-01-1996 12-01-1997
         4          2 01-04-1997 02-04-1997
         6          3 12-01-1996 12-01-1996
         8          3 04-08-2001 15-08-2001

6 rows selected.

SQL> savepoint inicio_subtransacao;
```

```
Savepoint created.
SQL> insert into solic_compra
 2 (num_solic
 3 ,dat_emiss
 4 ,dat_previsao
 5 ,cod_depto)
 6 values
 7 (9
 8 ,sysdate + 2
 9 ,sysdate + 10
 10 ,6);

1 row created.

SQL> select * from solic_compra;

NUM_SOLIC COD_DEPTO DAT_EMISS  DAT_PREVIS
--------- --------- ---------- ----------
        1         1 12-01-1996 12-02-1996
        2         3 12-12-1997 13-12-1997
        3         7 12-01-1996 12-01-1997
        4         2 01-04-1997 02-04-1997
        6         3 12-01-1996 12-01-1996
        8         3 04-08-2001 15-08-2001
        9         6 05-08-2001 13-08-2001

7 rows selected.

SQL> rollback to inicio_subtransacao;

Rollback complete.

SQL> select * from solic_compra;

NUM_SOLIC COD_DEPTO DAT_EMISS  DAT_PREVIS
--------- --------- ---------- ----------
        1         1 12-01-1996 12-02-1996
        2         3 12-12-1997 13-12-1997
        3         7 12-01-1996 12-01-1997
        4         2 01-04-1997 02-04-1997
        6         3 12-01-1996 12-01-1996
        8         3 04-08-2001 15-08-2001

6 rows selected.

SQL> commit;
```

```
Commit complete.
SQL> select * from solic_compra;

NUM_SOLIC COD_DEPTO DAT_EMISS  DAT_PREVIS
--------- --------- ---------- ----------
        1         1 12-01-1996 12-02-1996
        2         3 12-12-1997 13-12-1997
        3         7 12-01-1996 12-01-1997
        4         2 01-04-1997 02-04-1997
        6         3 12-01-1996 12-01-1996
        8         3 04-08-2001 15-08-2001

6 rows selected.
```

19 INCLUINDO LINHAS EM UMA TABELA

Sintaxe:
```
INSERT INTO nome_tabela
        ([nome_coluna_1]
        ,[nome_coluna_2]
        ,[nome_coluna_n])
    VALUES
        ([valor_coluna_1]
        ,[valor_coluna_2]
        ,[valor_coluna_n])
```

Observação:
Pode-se omitir a relação de colunas somente quando forem informados valores para todas as colunas.

➔ Exemplo:

```
SQL> insert into solic_compra
        2 (num_solic
        3 ,dat_emiss
,dat_previsao
,deptno)
        6 val
VALues
(1
,sysdate
,sysdate + 15
,10);

1 row created.
```

20 RECUPERANDO DADOS DE VÁRIAS TABELAS

JOIN

Exibe em uma mesma dupla de resultado dados que estejam em tabelas diferentes.

EQUI-JOIN OU RELAÇÃO DIRETA

Acontece quando há relacionamento de colunas entre duas ou mais tabelas.

➔ Exemplo:

```
SQL> select dname
        ,ename
        from emp, dept
        where  emp.deptno = dept.deptno
        order by dname, ename;

DNAME         ENAME
-------------- ----------
ACCOUNTING     CLARK
ACCOUNTING     KING
ACCOUNTING     MILLER
RESEARCH       ADAMS
RESEARCH       FORD
RESEARCH       JONES
RESEARCH       SCOTT
RESEARCH       SMITH
SALES          ALLEN
SALES          BLAKE
SALES          JAMES
SALES          MARTIN
SALES          TURNER
SALES          WARD
```

NON-EQUI-JOIN OU RELAÇÃO INDIRETA

Acontece quando não existe nenhuma coluna que se relacione diretamente com outra coluna de uma tabela diferente.

➔ Exemplo:

```
SQL> select e.ename
        2 ,e.sal
        3 ,s.grade
        4 from emp e, salgrade s
        5 where e.sal between s.losal and s.hisal;

ENAME          SAL    GRADE
---------- --------- ---------
SMITH          800          1
ADAMS          1100         1
WARD           1250         2
MARTIN         1250         2
ALLEN          1600         3
TURNER         1500         3
JONES          2975         4
FORD           3000         4
KING           5000         5
```

OUTER JOIN

Quando uma das tabelas não contém todos os elementos existentes na outra, alguns elementos desta não serão recuperados. Para que isto aconteça, utiliza-se o (+) o lado da tabela onde os elementos estão faltando.

➔ Exemplo:

```
SQL> insert into solic_compra values
        (12, null, sysdate, sysdate + 30);

1 row created.

SQL>  select d.cod_depto
        2 ,d.dsc_departamento
        3 ,count(s.num_solic)
        4 from depto d
        5 ,solic_compra s
        6 where d.cod_depto = s.cod_depto (+)
        7 group by d.cod_depto, d.dsc_departamento;
```

OBS.: (+) → onde não existem dados.

```
COD_DEPTO DSC_DEPARTAMENTO              COUNT(S.NUM_SOLIC)
--------- ------------------------------ ------------------
        1 Administração                  1
        2 Informática                    1
        3 Recursos Humanos               2
        4 Produção                       0
        5 Compras                        0
        6 Vendas                         0
        7 Marketing                      1

7 rows selected.
```

TEORIA DOS CONJUNTOS

UNION

A união de duas relações é o conjunto de todas as linhas que estão em uma ou outra relação, ignorando as duplicadas, ou seja, retorna à união de dois select's, ignorando as linhas duplicadas.

➔ Exemplo:

```
SQL> select * from emp
        2 where sal < 1000
        3 union
        4 select * from emp
        5 where sal > 2000;

  EMPNO ENAME      JOB              MGR HIREDATE         SAL          COMM
DEPTNO
--------- ---------- --------- --------- --------- --------- --------- ---------
   7369 SMITH        CLERK           7902 17-DEC-80       800         20
   7566 JONES        MANAGER         7839 02-APR-81      2975         20
   7698 BLAKE        MANAGER         7839 01-MAY-81      2850         30
   7782 CLARK        MANAGER         7839 09-JUN-81      2450         10
   7788 SCOTT        ANALYST         7566 19-APR-87      3000         20
   7839 KING         PRESIDENT       17-NOV-81           5000         10
   7900 JAMES        CLERK           7698 03-DEC-81       950         30
   7902 FORD          ANALYST        7566 03-DEC-81      3000         20
```

INTERSECT

A interseção é o conjunto de todas as linhas que estão simultaneamente em ambas as relações, ou seja, retorna à interseção de dois select's.

➔ Exemplo:

```
SQL> select * from emp
  2  where sal > 1000
  3  intersect
  4  select * from emp
  5  where sal < 2000;

  EMPNO ENAME      JOB          MGR HIREDATE       SAL       COMM     DEPTNO
--------- ---------- --------- --------- --------- --------- --------- ---------
   7499 ALLEN      SALESMAN       7698 20-FEB-81       1600        300         30
   7521 WARD       SALESMAN       7698 22-FEB-81       1250         50         30
   7654 MARTIN     SALESMAN       7698 28-SEP-81       1250       1400         30
   7844 TURNER     SALESMAN       7698 08-SEP-81       1500          0         30
   7876 ADAMS      CLERK          7788 23-MAY-87        100                    20
   7934 MILLER     CLERK          7782 23-JAN-82       1300                    10
```

MINUS

A diferença é o conjunto de todas as linhas que estão em apenas uma das relações, ou seja, retorna à subtração de dois select's.

➔ Exemplo:

```
SQL> select * from emp
 2 where sal > 3000
 3 minus
 4 select * from emp
 5 where sal < 1000;

  EMPNO ENAME      JOB          MGR HIREDATE       SAL       COMM     DEPTNO
--------- ---------- --------- --------- --------- --------- --------- ---------
   7839 KING       PRESIDENT           17-NOV-81       5000                    10
```

Regras para Utilizar Union, Intersect e Minus:

A cláusula Select deve selecionar o mesmo número de colunas.
As colunas correspondentes devem ser do mesmo tipo de dado.
As linhas duplicadas são automaticamente eliminadas.
Os nomes das colunas do primeiro select é que aparecem no resultado.
A cláusula order by deve aparecer no final do comando.
A cláusula order by somente pode ser usada indicando o número da coluna.
Os operadores Union, Intersect e Minus podem ser utilizados em subqueries.
As cláusulas Select são executadas de cima para baixo.

21 SUBQUERY

É uma cláusula select que é aninhada com outra cláusula select e que retorna a um resultado intermediário.

➔ Exemplo:

```
SQL> select ename from emp
 2 where sal = ( select max(sal) from emp);

ENAME
-----
KING
```

SUBQUERIES QUE RETORNAM MAIS DE UMA LINHA

No surgimento de uma subquery, na qual o resultado retorna a mais de uma linha da tabela, utilizar o operador IN ou EXISTS.

➔ Exemplo:

```
 SQL> select dname from dept
 2 where deptno in (select deptno from solic_compra);

DNAME
----------
ACCOUNTING
```

SUBQUERY NA CLÁUSULA HAVING

Relembrando, o where refere-se a uma linha da tabela e o having a um grupo de linhas especificadas na cláusula group by.

➔ Exemplo:

```
SQL> select job, avg(sal) from emp
 2 having avg(sal) > (select avg(sal) from emp
 3 where deptno = 10)
 4 group by job;

JOB       AVG(SAL)
--------- ---------
ANALYST       3000
PRESIDENT     5000
```

22 ATUALIZANDO DADOS COM O SQL*PLUS

Transações

Em uma transação pode-se executar inclusões, seleções, alterações e exclusões. Quando a transação é finalizada com o comando COMMIT, estas modificações ficam disponibilizadas para todos os usuários. Às vezes as alterações de um determinado usuário podem distorcer o resultado das seleções feitas em uma sessão. Por exemplo, em caso de relatórios que envolvam números.

É possível a um usuário ter uma visão congelada do Banco de Dados em um determinado momento, ou seja, desconsiderando quaisquer modificações que, por ventura, venham a ser feitas por outros usuários.

O comando SET TRANSACTION determina o início deste processo, que é encerrado pelo COMMIT ou ROLLBACK. Aloca a tabela e os outros usuários não podem manipulá-la, têm que esperar.

Sintaxe: SET TRANSACTION (READ ONLY| READ WRITE)

Atualizando dados de uma tabela

UPDATE

Através deste comando é possível atualizar uma ou mais colunas de uma tabela.

Sintaxe:
```
UPDATE nome da tabela
SET nome da coluna = valor [,
nome da coluna = valor]
WHERE condições;
```

➔ Exemplo:

```
SQL>  update emp set sal = sal * 1.2
 2 where sal < 1000;

 2 rows updated.
```

DELETE

Permite excluir uma ou mais linhas de uma tabela.

Sintaxe:
```
DELETE [FROM] nome da tabela
WHERE condições;
```

➔ Exemplo:

```
SQL> delete emp where sal < 1000;

1 row deleted.
```

23 ELIMINANDO TABELAS

É possível eliminar tabelas do Banco de Dados utilizando o comando DROP TABLE. Este comando elimina a tabela inclusive se ela tiver dados. Deve-se levar em conta os relacionamentos referentes a esta tabela. Por exemplo, não conseguimos eliminar a tabela DEPTO se ela estiver relacionada com a tabela SOLIC_COMPRA (A tabela SOLIC_COMPRA possui uma foreing key de depto).

Sintaxe: DROP TABLE nome da tabela;

➔ Exemplo:

```
SQL> drop table produto;

Table dropped
```

Renomeando tabelas

Pode-se renomear o nome de uma tabela ORACLE através do comando RENAME.

Sintaxe: RENAME nome tabela antiga TO nome tabela nova

➔ Exemplo:

```
SQL> rename produto to produto_2;

Table renamed.
```

Criando uma tabela a partir de outra tabela

Sintaxe:

```
CREATE TABLE nome da tabela
(nome da coluna [ restrição] [,
nome da coluna [ restrição]] [,restrições])
AS
SELECT
```

➔ Exemplo:

```
SQL> create table backup_produto as select * from produto;

Table created.
```

Incluindo linhas em uma tabela a partir de outra tabela:

Sintaxe: INSERT INTO nome da tabela
[(nome da coluna [,
nome da coluna])]
SELECT

➔ Exemplo:

```
SQL> insert into backup_produto select * from produto;

10 rows created.
```

24 VIEWS

Às vezes, uma tabela possui colunas que não devem ser visíveis para todos os usuários. Desta forma é necessário criar uma visão parcial da tabela. Outra situação é reunir várias tabelas em uma view para o usuário ter uma visão consolidada dos dados.

Uma view é uma tabela lógica que não ocupa espaço no Banco de Dados, pode ser composta de linhas de uma ou mais tabelas, agrupadas ou não.

CRIANDO UMA VIEW

Sintaxe: CREATE VIEW nome da view
[nome da coluna [,nome da coluna]]
AS
SELECT

➔ Exemplo

```
SQL>  create view visao_salario as
        select dname
        ,ename
        ,sal
        from   emp
        ,dept
        where  emp.deptno = dept.deptno
        and    sal > 3000;

View created.

SQL> select * from visao_salario;

DNAME         ENAME          SAL
------------- ---------- ---------
ACCOUNTING    KING          5000
```

ELIMINANDO UMA VIEW

Sintaxe: DROP VIEW nome da view

➔ Exemplo

```
SQL> drop view visao_salario;

View dropped.
```

25 MANIPULAÇÃO DE SEQUÊNCIAS

Sequência é uma estrutura criada no Banco de Dados que retorna para cada acesso um número diferente. Ela pode ser utilizada para fornecer, sequencialmente e sem repetição, valores a serem armazenados em códigos que formam a chave primária de uma tabela.

Sintaxe: CREATE SEQUENCE nome da sequência
[START WITH valor inicial]
[INCREMENT BY incremento]
[MAXVALUE valor máximo da sequência | NONMAXVALUE]
[MINVALUE valor mínimo da sequência | NONMINVALUE]
[CYCLE | NOCYCLE]
[CACHE número | NOCACHE]

Para recuperar o valor de uma sequência, deve-se utilizar as funções NEXTVAL (Próximo Valor) ou CURRVAL (Valor Corrente). Exceto o valor inicial, todos os parâmetros de uma sequência podem ser alterados através do comando ALTER SEQUENCE.

CYCLE | NOCYCLE: Determina se a sequência é cíclica ou não. Se for cíclica, quando atingir o valor máximo voltará ao inicial.
CACHE | NOCACHE: Determina se os valores das sequências devem ser pré-colocados em memória ou não.

CRIANDO UMA SEQUÊNCIA

```
SQL> create sequence conta_solic
 2 start with 1
 3 increment by 1;

Sequence created.
```

Observação:
A função CURRVAL recupera o último valor que é incrementado pela função NEXTVAL. Pode ser utilizado na recuperação da última sequência gerada de notas fiscais de uma empresa. Neste caso, portanto, a utilização da função CURRVAL será válida quando o arquivo de notas fiscais conter problemas.

```
SQL> select conta_solic.nextval from dual;

  NEXTVAL
---------
       1

SQL> select conta_solic.nextval from dual;

  NEXTVAL
---------
       2

SQL> select conta_solic.currval from dual;

  CURRVAL
---------
       2
```

```
SQL> select conta_solic.currval from dual;

  CURRVAL
---------
        2
```

ELIMINANDO UMA SEQUÊNCIA

```
SQL> drop sequence conta solic;
```

Sequence dropped

LEITURA COMPLEMENTAR

O GERENCIAMENTO DE BANCO DE DADOS

Os Bancos de Dados são coleções de informações que se relacionam para criar um significado dentro de um contexto computacional. Operados por Sistemas Gerenciadores de Banco de Dados (SGBDs), como Oracle, MySQL, SQL e PostgreSQL, os BDs são a base para a maioria das aplicações utilizadas pelas empresas, como os CRM, ERP, SCM, Service Desk ou qualquer solução que tenha como base um conjunto de informações que possam ser cruzadas, analisadas, filtradas ou tenham um objetivo específico para o seu armazenamento.

Manter um Banco de Dados com informações atualizadas é fator determinante para a assertividade dos processos de uma empresa. Além disso, outros problemas operacionais precisam ser observados para se alcançar um alto nível de competitividade. Problemas como a falta de recursos de hardware e software, definição e aplicação de uma estratégia de backup, ausência de um profissional dedicado (DBA), a falta de manutenção e de métricas de controle minam os processos das empresas e impactam em praticamente todas as áreas.

O monitoramento/gerenciamento do Banco de Dados é vital para o controle destes problemas. Além de identificá-los, é preciso descobrir qual a causa raiz destes problemas para poder tratá-los ou antecipá-los, já em um processo mais avançado de governança de TI. Em nossa experiência como fabricante de software para gerenciamento de TI, o Banco de Dados sempre foi umas das principais demandas, principalmente por afetar a maioria dos processos organizacionais. Uma vez que isto se torna um problema, pode gerar anos de dor de cabeça para os gestores do negócio que não possuem métricas de controle e processos/metodologias bem definidos.

A OpServices possui uma década experiência em monitoração de Banco de Dados através do **OpMon**. Além da possibilidade de criação de catálogos de serviços, onde é possível descobrir os processos e os serviços afetados por alguma indisponibilidade no Banco de Dados, enxergamos nos dashboards de negócio (ver OpMon Dashboards) um modo de transformar dados antes exclusivos da área de TI em dados utilizados pelos decisores do negócio para a tomada de decisão. Neste caso, uma visão de como os problemas originados em um BD pode impactar o faturamento, por exemplo. Desde que os gestores conheçam muito bem estes processos, é claro.

Já na esfera das oportunidades de negócio, a análise de grandes volumes de informações brutas (Big Data) é uma oportunidade para a resolução de uma grande quantidade de problemas em escala. O Big Data é um assunto que se tornou pauta de qualquer lista de tendências em tecnologia da informação para os próximos anos. Ele é baseado na metodologia dos 5V's: velocidade, volume, variedade, veracidade e valor. Um gerenciamento adequado do banco de dados impacta diretamente em todos estes itens. Os profissionais que visualizam essa oportunidade certamente agregam muito valor para o negócio da sua empresa.

FONTE: Disponível em: <http://www.opservices.com.br/importancia-gerenciamento-banco-dados/>

RESUMO DO TÓPICO 1

Neste capítulo você aprendeu que:

- SELECT é um comando de recuperação de dados.
- INSERT, UPDATE E DELETE são comandos de manipulação de dados.
- CREATE, ALTER, DROP, RENAME E TRUNCATE são comandos de definição de objetos.
- COMMIT, ROLLBACK E SAVEPOINT são comandos de controle de transações.
- GRANT E REVOKE são comandos de controle de acesso.
- SQL é uma linguagem declarativa e que não é possível criar programas inteiros com ela.
- SQL utiliza padrão ANSI, podendo ser usada em qualquer SGBD relacional.
- PLSQL é uma linguagem imperativa, permitindo que se criem programas inteiros em sua estrutura.
- **O UNION** combina conjuntos de linhas, eliminando as linhas duplicadas (menos performático que o UNION ALL).
- O **UNION ALL c**ombina conjuntos de linhas sem eliminar as duplicadas.
- O **INTERSECT** inclui linhas comuns em ambas as queries, eliminando as duplicadas.
- O **MINUS** retorna o que existe no primeiro conjunto de linhas menos o segundo conjunto de linhas (as linhas que têm na primeira tabela, que não têm na segunda tabela, ou seja, retornam as linhas que existem unicamente no primeiro SELECT).
- Uma subquery pode ser usada em um comando CREATE TABLE para criar e já popular uma tabela, técnica essa chamada de CTAS ou CREATE TABLE AS SELECT. Sua sintaxe é: CREATE TABLE MINHA_TABELA_COPIA AS SELECT * FROM MINHA_TABELA_ORIGINAL

- Subqueries podem ser incorporadas em comandos de INSERT e UPDATE para mover grandes quantidades de dados, ou alterar grandes quantidades de dados em um único SQL.

- Existem alguns tipos de subqueries, cada um com suas vantagens/desvantagens.

NOTA 1 – Caso a subquery retorne 0 linhas, então o valor retornado pela subquery é NULL.
NOTA 02 – É possível ter subqueries no WHERE, HAVING e GROUP BY.

RESUMO DOS PRINCIPAIS COMANDOS

Controlando o Acesso aos Dados: DCL - DATA CONTROL LANGUAGE

GRANT SELECT, INSERT, UPDATE, DELETE ON EMPREGADOS TD USER12;
REVOKE DELETE ON EMPREGADOS FROM USER12;

Gerenciando Índices: DDL - DATA DEFINITION LANGUAGE

CREATE INDEX EMPREGADOS_IDX ON EMPREGADOS(NOME);

CREATE UNIQUE INDEX EMPREGADOS_IDX ON EMPREGADOS(NOME);

CREATE INDEX EMPREGADOS_IDX ON EMPREGADOS(NOME, PAGAMENTO);

DROP INDEX EMPREGADOS IDX;

Gerenciando Visões (views): DDL - DATA DEFINITION LANGUAGE

```
CREATE VIEW MAIOR_PAGAMENTO_EMPREGADOS AS
SELECT FROM EMPREGADOS
WHERE PAGAMENTO > 150;

CREATE VIEW NOMES AS
SELECT NOME FROM EMPREGADOS;

DROP VIEW NOMES;
```

Funções de Agregação:

COUNT - Retorna o número de linhas
SUM - Retorna a soma de uma coluna específica
AVG - Retorna o valor médio de uma coluna específica
MAX - Retorna o valor máximo de uma coluna específica

MIN - Retorna o valor mínimo de uma coluna específica

Exemplos:
SELECT AVG(PAGAMENTO) FROM EMPREGADOS;

SELECT COUNT(*) FROM EMPREGADOS;

Operadores Lógicos:

IS NULL
BETWEEN
IN
LIKE
EXISTS
UNIQUE
ALL and ANY

Exemplos:
SELECT * FROM EMPREGADOS
WHERE PAGAMENTO BETWEEN 100 AND 150;

SELECT NOME, PAGAMENTO
FROM EMPREGADOS
WHERE EID IN ('1111', '2222', '3333');

Negando Condições com o Operador NOT:

NOT EQUAL
NOT BETWEEN
NOT IN
NOT LIKE
NOT EXISTS
NOT UNIQUE

Exemplos:
SELECT * FROM EMPREGADOS
WHERE PAGAMENTO NOT BETWEEN 100 AND 150;

SELECT NOME, PAGAMENTO
FROM EMPREGADOS
WHERE EID NOT IN ('1111', '2222', '3333');

SELECT NOME FROM EMPREGADOS
WHERE NOME NOT LIKE 'S%';

AUTOATIVIDADE

1. Crie a base de dados abaixo:

```
CREATE TABLE depto
(cod_depto            number(7)    not null
,dsc_departamento     varchar2(30) not null
,constraint depto_pk primary key (cod_depto));

CREATE TABLE produto
(cod_produto          number(7)    not null
,dsc_produto          varchar2(30)  not null
,dsc_unidade          varchar2(10)
,vlr_unit             number(12,02) not null
,constraint produto_pk primary key (cod_produto));

CREATE TABLE fornecedor
(cod_fornec           number(7)    not null
,dsc_razao_social     varchar2(40)  not null
,num_cgc              number(15)   not null
,num_cep              number(8)    not null
,cod_cidade           number(4)    not null
,dsc_endereco         varchar2(40)  not null
,cod_tipo_fornec      varchar2(1)
,tipo_fornec          number(5)
,num_ins_est          number(11)
,num_fone             number(13)
,num_fax              number(11)
,num_telex            number(11)
,nom_contato          varchar2(30)
,dat_nasc_contato     date
,constraint fornec_pk primary key (cod_fornec));

CREATE TABLE solic_compra
(num_solic            number(7)    not null
,cod_depto            number(7)    not null
,dat_emiss            date         not null
,dat_previsao         date         not null
,constraint solic_pk primary key (num_solic)
,constraint solic_depto_fk foreign key (cod_depto)
    references depto (cod_depto));
```

```
CREATE TABLE pedido
(num_pedido          number(7)    not null
,cod_fornec          number(7)    not null
,dat_emiss           date         not null
,dat_entreg          date         not null
,constraint pedido_pk primary key (num_pedido)
,constraint pedido_fornec_fk foreign key (cod_fornec)
    references fornecedor (cod_fornec));


CREATE TABLE item_solic
(num_solic           number(7)    not null
,cod_produto         number(7)    not null
,qtd_solic           number(7)    not null
,constraint item_solic_pk primary key (num_solic, cod_produto)
,constraint item_solic_produto_fk foreign key (cod_produto)
    references produto (cod_produto)
,constraint item_solic_solic_fk foreign key (num_solic)
    references solic_compra (num_solic));

CREATE TABLE item_pedido
(num_pedido          number(7)    not null
,cod_produto         number(7)    not null
,num_solic           number(7)    not null
,qtd_pedido          number(7)    not null
,vlr_unit            number(7,2)  not null
,constraint item_ped_pk primary key (num_solic, cod_produto, num_pedido)
,constraint item_ped_item_solic_fk foreign key (num_solic, cod_produto)
    references item_solic (num_solic, cod_produto)
,constraint item_ped_pedido_fk foreign key (num_pedido)
    references pedido (num_pedido));
```

2. Insira os seguintes dados na base:

```
insert into depto values (1,'Administração');
insert into depto values (2,'Informática');
insert into depto values (3,'Recursos Humanos');
insert into depto values (4,'Produção');
insert into depto values (5,'Compras');
insert into depto values (6,'Vendas');
insert into depto values (7,'Marketing');
```

```
insert into produto values (1,'Parafuso','Unidade',.10);
insert into produto values (2,'Martelo','Unidade',1.99);
insert into produto values (3,'Graxa','Lata',.30);
insert into produto values (4,'Furadeira Bosch','Unidade',75.60);
insert into produto values (10,'Esquadro',null,15);
insert into produto values (11,'Alicate','Unidade',3);
insert into produto values (12,'Betoneira','Unidade',150);
insert into produto values (5,'Chave de Fenda','Unidade',1.4);
insert into produto values (6,'Cano de PVC','Metro',0.90);
insert into produto values (13,'Massa de Vedação','Quilo',4);

insert into fornecedor values (1,'Informática Blumenau',1.235E+09,88301150,
1,'Rua Itaiópolis',1,1,6534321,3230101,null,null,'Fernando','12/09/1975');
insert into fornecedor values (2,'Armazém das Ferragens',65464,4654654,
1,'Rua São Benedito',4,2,1234234,2228989,null,null,'Edenilsom
Capestrim','12/01/1946');
insert into fornecedor values (3,'Casas da Água',54654654,65465465,
1,'Rua Capião Euclides de Castro',1,1,12112,1211212,null,null,'Jandira
Aspertes','12/10/1973');

insert into solic_compra values (1,1,'12/01/1996','12/02/1996');
insert into solic_compra values (2,3,'12/12/1997','13/12/1997');
insert into solic_compra values (3,7,'12/01/1996','12/01/1997');
insert into solic_compra values (4,2,'01/04/1997','02/04/1997');
insert into solic_compra values (6,3,'12/01/1996','12/01/1996');

insert into pedido values (2,1,'12/01/1996','12/03/1997');
insert into pedido values (3,2,'24/01/1997','30/01/1997');
insert into pedido values (4,2,'12/01/1997','12/01/1997');
insert into pedido values (5,3,'15/02/1997','20/02/1997');
insert into pedido values (6,1,'12/09/1997','20/09/1997');
insert into pedido values (7,2,'23/01/1997','24/01/1997');
insert into pedido values (8,3,'15/01/1997','15/01/1997');

insert into item_solic values (1,2,3);
insert into item_solic values (1,3,2);
insert into item_solic values (1,4,4);
insert into item_solic values (2,2,2);
insert into item_solic values (2,4,1);
insert into item_solic values (2,6,5);
insert into item_solic values (3,1,2);
insert into item_solic values (3,5,1);
insert into item_solic values (4,1,100);
```

```
insert into item_solic values (4,2,1);
insert into item_solic values (4,3,1);
insert into item_solic values (4,4,1);
insert into item_solic values (4,5,1);
insert into item_solic values (4,6,10);
insert into item_solic values (6,2,200);

insert into item_pedido values (2,2,1,2,50.78);
insert into item_pedido values (3,2,1,3,250.56);
insert into item_pedido values (3,3,1,2,536.32);
insert into item_pedido values (4,4,1,4,1050.23);
insert into item_pedido values (4,3,1,3,78.23);
insert into item_pedido values (4,2,1,4,12.33);
insert into item_pedido values (5,2,2,2,545.12);
insert into item_pedido values (5,4,2,2,256.66);
insert into item_pedido values (5,6,2,6,89.2);
insert into item_pedido values (6,1,3,2,88.26);
insert into item_pedido values (6,5,3,2,257.12);
insert into item_pedido values (7,1,4,150,78.15);
insert into item_pedido values (7,2,4,2,4056.56);
insert into item_pedido values (7,3,4,2,455.26);
insert into item_pedido values (7,5,4,3,789.65);
insert into item_pedido values (7,6,4,15,798.54);
insert into item_pedido values (7,4,4,2,10.12);
insert into item_pedido values (8,2,6,200,4998.23);
```

3. Resolva a lista de exercícios utilizando a base proposta:

a) Mostrar todos os itens de solicitação do produto 5.

b) Mostrar todos os itens de solicitação em que a quantidade solicitada é de até 3 unidades.

c) Mostrar todos os itens de solicitação em que a quantidade solicitada é acima de 5 unidades.

d) Mostrar todas as descrições de produto que não são medidas por ‚Unidade'.

e) Mostrar código e descrição dos produtos em que a unidade de medida é nula.

f) Mostrar a descrição dos departamentos 3, 4 e 7.

g) Mostrar o número dos pedidos emitidos em janeiro de 1997.

h) Mostrar todas as informações dos fornecedores em que a razão social contém a string ‚err'.

i) Mostrar código e descrição dos produtos que foram pedidos pelo menos uma vez.

j) Mostrar código e descrição dos departamentos que fizeram pelo menos uma solicitação de compra.

k) Mostrar todas as informações dos produtos que são medidos por Lata ou que custam até R$ 1,00.

l) Mostrar todas as informações dos produtos que são medidos por Unidade e custam mais de R$ 50,00.

m) Mostrar todas as informações dos produtos que são medidos por Unidade e custam mais de R$ 50,00 ou que são medidas por Metro.

n) Mostrar o código dos produtos solicitados, além da quantidade média e total solicitada.

o) Mostrar a quantidade pedida, aberta por fornecedor e produto.

p) Mostrar para cada produto pedido as seguintes informações:

1) Descrição.
2) Quantidade de pedidos.
3) Quantidade pedida do produto.
4) Valor total pedido.
5) Menor preço unitário.
6) Maior preço unitário.
7) Preço médio unitário praticado.

q) Mostrar a quantidade de produtos solicitada e a média de produtos por solicitação, aberto por descrição do departamento, mês (e ano) de emissão da solicitação e código do produto.

r) Montar o seguinte texto (utilizar concatenação) sobre os produtos pedidos:

1) A quantidade do pedido.
2) A unidade de medida, sendo que:

2.1) Unidade deve ser representada por peça.
2.2) Metro deve ser representado por m.
2.3) Não deve ser levado em conta maiúsculo/minúsculo (Metro = metro METRO).

3) Descrição do produto.
4) Custo unitário.
5) Seguindo o exemplo.

s) Mostrar a razão social dos fornecedores que entregaram um pedido mais de 7 dias após a emissão do pedido. Mostrar também quantas vezes isso ocorreu.

t) Mostrar a razão social dos fornecedores e a quantidade de pedidos feita ao fornecedor. Mostrar também quantos pedidos foram entreges mais de 7 dias após a emissão.

TÓPICO 2

PLSQL

1 INTRODUÇÃO

A linguagem PL/SQL é uma extensão à linguagem SQL, portanto, todos os comandos SQL podem ser utilizados em PL/SQL. O PL/SQL é uma linguagem procedural e estruturada, com hierarquia de comandos e fluxo linear de execução.

O Banco de Dados possui mecanismos próprios que podem ser utilizados em favor do desenvolvedor. Cada banco de dados possui um conjunto específico de comandos que definem a linguagem de programação do Banco de Dados. No caso do Oracle, a linguagem é o PL/SQL, o SQL Server possui o Transact-SQL, o DB2 possui sua própria linguagem de programação, o PostGreSQL possui diversas extensões que podem ser utilizadas como linguagem de programação e o MySQL lançou sua mais recente versão com a possibilidade de programar o servidor. Cada Banco de Dados é único sob este aspecto, mas todos trabalham sobre os mesmos conceitos. É possível criar módulos programáveis, como funções, procedimentos, objetos, pacotes, gatilhos etc. Em todos os casos, há um engine responsável pela integração e execução dos módulos no servidor de Banco de Dados.

Desta forma, os tópicos que serão trabalhados serão:

- Fundamentos de programação de banco de dados.
- Aspectos avançados da programação: ALGORITMOS E CURSORES.

2 VANTAGENS DO PL/SQL

- Linguagem procedural: loops, decisões, atribuições etc.
- Aumento de performance: vários blocos de comandos enviados para a base de uma só vez.
- Incremento da Produtividade: adaptação a outras ferramentas, como Forms e Reports.
- Portabilidade: aplicações escritas em PL/SQL podem ser executadas em qualquer plataforma, desde que contenham a base Oracle instalada.

- Integração com o RDBMS Oracle: a declaração %TYPE, por exemplo, permite trazer definições de tipos de dados da base Oracle para variáveis do código PL/SQL.
- Um bloco de PL/SQL pode ser executado através do SQL*PLUS, do SQL*Forms, do SQL*Menu ou ainda dentro de qualquer utilitário PRO*Oracle (Pro*C, Pro*Pascal etc.)

3 ESTRUTURA DE UM BLOCO PL/SQL

Um bloco PL/SQL é basicamente composto de uma área de declaração, uma área de comandos e uma área de tratamento de exceções. Todo e qualquer programa escrito em PL/SQL deve sempre obedecer a uma estrutura de blocos de identação, hierarquicamente definidos:

```
DECLARE
 'Declarações'
BEGIN
 'Estruturas executáveis e outros blocos PL/SQL'
EXCEPTION
 'Tratamento de exceções"
END;
```

Declarações

Local onde são especificados os dados utilizados no bloco PL/SQL. Os dados podem ser:

- Variáveis.
- Constantes.
- Cursores.
- Exceções.
- Estruturas.
- Tabelas.

Todo objeto que for referenciado pelos comandos PL/SQL deve pertencer à base Oracle ou estar declarado na seção de declaração.

Estruturas executáveis e outros blocos PL/SQL

Local onde os comandos PL/SQL estruturados de maneira lógica são colocados. Ficam entre o BEGIN e o END.

Tratamento de exceções

Exceções são todos os erros e imprevistos que podem ocorrer durante a execução de um bloco PL/SQL. Quando o erro ou imprevisto ocorre, o PL/SQL abandona a área de comandos e procura na área de exceções o tratamento para a falha ocorrida.

Esta sessão pode ser omitida, sendo que o programa é abortado em caso de erro.

4 ESCOPO DE OBJETOS EM PL/SQL

Todos os objetos declarados em uma seção DECLARE são locais ao bloco a que ela pertence e globais aos blocos interiores a ele.

```
DECLARE
        A CHAR(20);
        B NUMBER(2);
BEGIN
        DECLARE
        A NUMBER(20);        /*A variável 'A' global não é vista aqui */
BEGIN
        comandos PL/SQL
END;

DECLARE
        B CHAR(2);    /*A variável 'B' global não é vista aqui */
BEGIN
        comandos PL/SQL
END;
EXCEPTION                              /*Se houver erro de execução  o fluxo é
                                                    desviado para cá */
        when ...

END;
```

5 VARIÁVEIS

Objetos que podem armazenar valores temporariamente, podendo ser atualizadas a qualquer momento. Antes de utilizadas, as variáveis deverão ser declaradas na seção DECLARE.

Sintaxe:
<nome_variável> <tipo> (<tamanho>[,decimais>]);

OBS.: <tipo> é o tipo de dados que será armazenado na variável, por exemplo, CHAR, NUMBER, DATE, BOOLEAN (True ou False).

ATRIBUIÇÃO DE VALORES ÀS VARIÁVEIS

Para associar um valor a uma variável, utiliza-se o comando de atribuição ':=', e finaliza-se com o ponto e vírgula.

Ex.: A:=2;
B:=to_number(C);

Pode-se também utilizar comando SQL para atribuir valores a variáveis.

Em PL/SQL, todo o comando SELECT pede a cláusula INTO para associar valores de colunas da base Oracle a variáveis PL/SQL.

Sintaxe: SELECT colunas
INTO variáveis
FROM tabelas
WHERE condições
etc.

OBS.: Para cada coluna selecionada deve existir uma variável associada. A coluna e a variável devem ser de tipos compatíveis.

➔ Exemplo

Criar um programa que mostre o salário do empregado acrescido de 30%.

SQL>ed exemplo1

```
set serveroutput on

DECLARE
    aux_salario number;
```

```
BEGIN
        select  sal * 1.3
        into    aux_salario
        from    emp
        where   empno = &prm_empno;
        --
        dbms_output.put_line('Salario de ' || to_char(aux_salario));
END;
/

Enter value for prm_empno: 7934
old  7:   where   empno = &prm_empno;
new  7:   where   empno = 7934;
Salario de 1690

PL/SQL procedure successfully completed.
```

Nesse exemplo, existem algumas instruções não discutidas

A instrução *set serveroutput on* serve para que as mensagens geradas na execução do bloco PL/SQL sejam exibidas na tela. As mensagens são geradas através do comando *dbms_output.put_line*, recebendo um parâmetro do tipo alfanumérico, que será exibido na tela.

Como o parâmetro é alfanumérico, valores de outros tipos (date e number) devem ser convertidos através da função *to_char* para serem exibidos.

O símbolo *&* representa um parâmetro a ser solicitado ao usuário no momento da execução. O nome do parâmetro é declarado logo após o símbolo *&*. Caso seja usado um número após o símbolo &, serve como parâmetro passado na linha de comando para o script SQL. Nesse caso, o primeiro valor corresponde a &1, o segundo &2, e assim sucessivamente.

%TYPE e %ROWTYPE

%TYPE: Faz com que uma variável ou constante assuma o formato dos valores de uma coluna da base de dados.

➔ Exemplo
Criar um programa que mostre o salário do empregado acrescido de 30%. Utilizar %TYPE para definição da variável.

SQL> ed exemplo2

```
set serveroutput on
```

```
DECLARE
        aux_salario emp.sal%type;

BEGIN
        select  sal * 1.3
        into    aux_salario
        from    emp
        where   empno = &prm_empno;
        --
        dbms_output.put_line('Salario de ' || to_char(aux_salario));
END;
/

Enter value for prm_empno: 7900
old   7:    where   empno = &prm_empno;
new   7:    where   empno = 7900;
Salario de 1482

PL/SQL procedure successfully completed.
```

%ROWTYPE: Cria um registro completo, utilizando-se as características de uma tabela da base ou as colunas retornadas de um cursor. É como se a estrutura da tabela fosse armazenada na variável.

➔ Exemplo
Criar um programa que mostre o nome do empregado e o seu salário acrescido de 30%.
Utilizar %ROWTYPE para definição das variáveis.

```
SQL> ed exemplo3

set serveroutput on

DECLARE
        aux_empregado emp%rowtype;
BEGIN
        select  *
        into    aux_empregado
        from    emp
        where   empno = &prm_empno;
        --
        dbms_output.put_line('Salario de '
                ||to_char(aux_empregado.sal * 1.3)
                ||' para '||aux_empregado.ename);
```

```
END;
/
Enter value for prm_empno: 7900
old  7:   where  empno = &prm_empno;
new  7:   where  empno = 7900;
Salario de 1482 para JAMES

PL/SQL procedure successfully completed.
```

6 CONSTANTES

As constantes são objetos que, uma vez declarados, não podem ter seus valores alterados. Portanto, a constante sempre recebe um valor no momento da declaração. Utiliza-se a palavra CONSTANT antes de se especificar o tipo.

➔ Exemplo

```
DECLARE
        pi  CONSTANT   number(12,10) := 3,14159;
BEGIN
        <outras declarações>
END;
```

6.1 CONTROLE CONDICIONAL

Frequentemente, é necessário tomar determinadas ações alternativas dependendo do resultado de determinadas operações ou de circunstâncias quaisquer. Para isso utiliza-se o comando IF.

Sintaxes:

1) IF condição THEN
ação
END IF;

2) IF condição THEN
        ação1
    ELSE
        ação2
    END IF;

3) IF condição1 THEN
ação1
ELSIF condição2 THEN
ação2
ELSE
ação3
END IF;

➔ Exemplo
Criar um programa que calcule o aumento do salário do empregado de acordo com a faixa salarial na qual ele se encontra: 1 → 10%, 2→ 20%, 3→ 30%, 4→ 40% e 5→ 50%.

SQL> ed exemplo4

```
set serveroutput on

DECLARE
        aux_salario emp.sal%type;
        aux_faixa   salgrade.grade%type;
BEGIN
        select  e.sal
                ,s.grade
        into    aux_salario
                ,aux_faixa
        from    emp e
                ,salgrade s
        where   e.sal between s.losal and s.hisal
        and     e.empno = &prm_empno;
        --
        if aux_faixa = 1 then
                 aux_salario := aux_salario * 1.1;
        elsif aux_faixa = 2 then
                 aux_salario := aux_salario * 1.2;
        elsif aux_faixa = 3 then
                 aux_salario := aux_salario * 1.3;
        elsif aux_faixa = 4 then
                aux_salario := aux_salario * 1.4;
        elsif aux_faixa = 5 then
                aux_salario := aux_salario * 1.5;
        end if;
        --
        dbms_output.put_line('Salario de ' ||to_char(aux_salario));
```

```
END;
/
Enter value for prm_empno: 7499
old  12:    and     e.empno = &prm_empno;
new  12:    and     e.empno = 7499;
Salario de 2080

PL/SQL procedure successfully completed.
```

7 CONTROLE ITERATIVO

LOOP
O comando LOOP é utilizado para executar uma relação de comandos até que a condição definida na saída se torne verdadeira.

Sintaxe:

```
LOOP
        Relação_de_Comandos
END LOOP;
```

Para interromper a execução de um loop, utiliza-se o comando EXIT. Este comando desvia o fluxo do programa para a primeira instrução após o próximo END LOOP. Um comando EXIT não pode ser utilizado fora de loops.

Sintaxe:

```
EXIT [WHEN <condição>];
```

➔ Exemplo 1

Criar um programa que calcule o aumento de salário do empregado, de acordo com a faixa salarial na qual ele se encontra: 1 → 10%, 2→ 20%, 3→ 30%, 4→ 40% e 5→ 50%, aplicando o aumento quantas vezes o usuário informar.

SQL> ed exemplo5

```
set serveroutput on

DECLARE
        aux_salario emp.sal%type;
        aux_faixa   salgrade.grade%type;
        aux_repete  number;
        aux_conta   number;
```

```
BEGIN
        aux_repete := &prm_repeticoes;
        aux_conta  := 0;
        --
        select  e.sal
                ,s.grade
        into    aux_salario
                ,aux_faixa
        from    emp e
                ,salgrade s
        where   e.sal between s.losal and s.hisal
        and     e.empno = &prm_empno;
        --
        loop
        if aux_conta >= aux_repete then
        exit;
        else
        aux_conta := aux_conta + 1;
        end if;
        --
        if aux_faixa = 1 then
         aux_salario := aux_salario * 1.1;
        elsif aux_faixa = 2 then
         aux_salario := aux_salario * 1.2;
        elsif aux_faixa = 3 then
         aux_salario := aux_salario * 1.3;
        elsif aux_faixa = 4 then
         aux_salario := aux_salario * 1.4;
        elsif aux_faixa = 5 then
         aux_salario := aux_salario * 1.5;
        end if;
        end loop;
        --
        dbms_output.put_line('Salario de ' ||to_char(aux_salario));
END;
/
Enter value for prm_repeticoes: 1
old   7:    aux_repete := &prm_repeticoes;
new   7:    aux_repete := 1;
Enter value for prm_empno: 7499
old  17:    and     e.empno = &prm_empno;
new  17:    and     e.empno = 7499;
Salario de 2080
PL/SQL procedure successfully completed.
SQL> /
```

```
Enter value for prm_repeticoes: 2
old   7:    aux_repete := &prm_repeticoes;
new   7:    aux_repete := 2;
Enter value for prm_empno: 7499
old  17:    and     e.empno = &prm_empno;
new  17:    and     e.empno = 7499;
Salario de 2704
PL/SQL procedure successfully completed.
```

WHILE

O Comando WHILE associa uma condição à execução das iterações, sendo esta condição avaliada a cada iteração. Uma nova iteração só é iniciada se a condição for verdadeira.

Sintaxe:

```
WHILE <condição> LOOP
        comandos;
END LOOP;
```

➔ Exemplo

Criar um programa que calcule o aumento de salário do empregado de acordo com a faixa salarial na qual ele se encontra: 1 → 10%, 2→ 20%, 3→ 30%, 4→ 40% e 5→ 50%, aplicando o aumento quantas vezes o usuário informar. Utilizar o comando WHILE.

SQL> ed exemplo6

```
set serveroutput on

DECLARE
        aux_salario emp.sal%type;
        aux_faixa   salgrade.grade%type;
        aux_repete  number;
        aux_conta   number;
BEGIN
        aux_repete := &prm_repeticoes;
        aux_conta  := 0;
        --
        select  e.sal
                ,s.grade
        into    aux_salario
                ,aux_faixa
        from    emp e
                ,salgrade s
```

```
        where   e.sal between s.losal and s.hisal
        and     e.empno = &prm_empno;
        --
        while aux_conta < aux_repete loop
        aux_conta := aux_conta + 1;
        --
        if aux_faixa = 1 then
                aux_salario := aux_salario * 1.1;
        elsif aux_faixa = 2 then
                aux_salario := aux_salario * 1.2;
        elsif aux_faixa = 3 then
                aux_salario := aux_salario * 1.3;
        elsif aux_faixa = 4 then
                aux_salario := aux_salario * 1.4;
        elsif aux_faixa = 5 then
                aux_salario := aux_salario * 1.5;
        end if;
        end loop;
        --
        dbms_output.put_line('Salario de ' ||to_char(aux_salario));
END;
/
Enter value for prm_repeticoes: 2
old   7:    aux_repete := &prm_repeticoes;
new   7:    aux_repete := 2;
Enter value for prm_empno: 7499
old  17:    and     e.empno = &prm_empno;
new  17:    and     e.empno = 7499;
Salario de 2704

PL/SQL procedure successfully completed.
```

FOR
Permite que o loop seja repetido em uma conhecida quantidade de iterações.

Sintaxe:
FOR <cont> IN [REVERSE] <min>..<max> LOOP
comandos;
END LOOP;

➔ Exemplo

Criar um programa que calcule o aumento de salário do empregado de acordo com a faixa salarial na qual ele se encontra: 1 → 10%, 2→ 20%, 3→ 30%, 4→40% e 5→ 50%, aplicando o aumento quantas vezes o usuário informar.

Utilizar o comando FOR.

```
SQL> ed exemplo7

set serveroutput on
DECLARE
        aux_salario emp.sal%type;
        aux_faixa   salgrade.grade%type;
        aux_repete  number;
        aux_conta   number;
BEGIN
        aux_repete := &prm_repeticoes;
        --
        select  e.sal
                ,s.grade
        into    aux_salario
                ,aux_faixa
        from    emp e
                ,salgrade s
        where   e.sal between s.losal and s.hisal
        and     e.empno = &prm_empno;
        --
        for aux_conta in 1 .. aux_repete loop
        if aux_faixa = 1 then
                aux_salario := aux_salario * 1.1;
        elsif aux_faixa = 2 then
                aux_salario := aux_salario * 1.2;
        elsif aux_faixa = 3 then
                aux_salario := aux_salario * 1.3;
        elsif aux_faixa = 4 then
                aux_salario := aux_salario * 1.4;
        elsif aux_faixa = 5 then
                aux_salario := aux_salario * 1.5;
        end if;
        end loop;
        --
  dbms_output.put_line('Salario de ' ||to_char(aux_salario));
END;
/
Enter value for prm_repeticoes: 2
old   7:    aux_repete := &prm_repeticoes;
new   7:    aux_repete := 2;
Enter value for prm_empno: 7499
old  16:    and     e.empno = &prm_empno;
new  16:    and     e.empno = 7499;
Salario de 2704
PL/SQL procedure successfully completed.
```

8 CURSORES

São estruturas criadas para selecionar várias linhas de resultado, ou seja, são comandos *select* que podem retornar mais de uma linha de resultado (ou nenhuma). Essa flexibilidade não é permitida através da sintaxe *select ... into ...* Dessa forma, a utilização de cursores garante a não ocorrência dos erros *ORA-01403: no data found* e *ORA-01422: exact fetch returns more than requested number of rows*.

CURSORES EXPLÍCITOS

São cursores utilizados para a manipulação de buscas que retornam mais de uma linha de resultado. Estes cursores devem ser declarados na seção DECLARE.

Sintaxe:
CURSOR <nome cursor> IS
<comando SELECT qualquer>
Antes de manipular um cursor, ele deve ser aberto através do comando OPEN.

Sintaxe:
OPEN <nome cursor>

➔ Exemplo

```
declare
        cursor deptos is
                select  deptno
                        ,dname
        from   dept;
begin
        open dept;
        <outras instruções>
end;
```

Para processar as linhas armazenadas no cursor, deve-se carregá-las uma a uma para variáveis declaradas no bloco PL/SQL. O comando FETCH é o encarregado de realizar esta cópia.

Sintaxe:
FETCH <nome cursor>
INTO <lista variaveis>

➔ Exemplo

```
declare
        aux_deptno   dept.deptno%type;
        aux_dname   dept.dname%type;

        cursor deptos is
        select deptno
                ,dname
        from   dept;
begin
        open deptos;
        fetch deptos
        into   aux_deptno
        ,aux_dname;
        <outras instruções>
end;
```

Quando um cursor não é mais necessário, deve-se fechá-lo para liberar a área utilizada por ele.

Sintaxe:
CLOSE <nome cursor>

Cada cursor definido explicitamente possui 4 atributos:

%NOTFOUND: Tipo boolean, retorna TRUE ou FALSE conforme o resultado do FETCH.
Se o último FETCH executado trouxe um registro retorna o valor FALSE, caso contrário, retorna o valor TRUE.

%FOUND: Oposto de %NOTFOUND.
Se o último FETCH executado trouxe um registro retorna o valor TRUE, caso contrário retorna o valor FALSE.

%ROWCOUNT: Retorna a quantidade de registros que já foram trazidos de um cursor através do FETCH.

%ISOPEN: Retorna TRUE se o cursor estiver aberto.

➔ Exemplo:

Criar um programa que calcule o aumento de salário de todos os empregados, de acordo com a faixa salarial na qual ele se encontra: 1 → 10%, 2→20%, 3→ 30%, 4→40% e 5→ 50%.

SQL> ed exemplo8

```
set serveroutput on

DECLARE
        aux_nome    emp.ename%type;
        aux_salario emp.sal%type;
        aux_faixa   salgrade.grade%type;
        --
        cursor empregados is
        select  e.ename
                ,e.sal
                ,s.grade
        from    emp e
                ,salgrade s
        where   e.sal between s.losal and s.hisal;
BEGIN
        open empregados;
        loop
        fetch empregados
 into  aux_nome
                ,aux_salario
        ,aux_faixa;
 exit when empregados%notfound;
        --
        if aux_faixa = 1 then
                aux_salario := aux_salario * 1.1;
        elsif aux_faixa = 2 then
                aux_salario := aux_salario * 1.2;
        elsif aux_faixa = 3 then
                aux_salario := aux_salario * 1.3;
        elsif aux_faixa = 4 then
                aux_salario := aux_salario * 1.4;
        elsif aux_faixa = 5 then
                aux_salario := aux_salario * 1.5;
        end if;
        --
  dbms_output.put_line('Salario de ' ||to_char(aux_salario)
                ||' para '||aux_nome);
 end loop;
 close empregados;
```

```
END;
/
Salario de 880 para SMITH
Salario de 1210 para ADAMS
Salario de 1045 para JAMES
Salario de 1500 para WARD
Salario de 1500 para MARTIN
Salario de 1560 para MILLER
Salario de 2080 para ALLEN
Salario de 1950 para TURNER
Salario de 4165 para JONES
Salario de 3990 para BLAKE
Salario de 3430 para CLARK
Salario de 4200 para SCOTT
Salario de 4200 para FORD
Salario de 7500 para KING

PL/SQL procedure successfully completed.
```

CURSORES IMPLÍCITOS

O Banco de Dados implicitamente abre um cursor para processar cada estrutura SQL que não estiver associada a um cursor explícito.

Não é possível executar operações do tipo OPEN, FETCH e CLOSE em um cursor implícito, porém é possível avaliar o resultado de sua operação através dos atributos SQL%NOTFOUND, SQL%ROWCOUNT, SQL%FOUND e SQL%ISOPEN.

UTILIZANDO A INSTRUÇÃO FOR JUNTO AOS CURSORES EXPLÍCITOS

➔ Exemplo

Criar um programa que calcule o aumento de salário de todos os empregados, de acordo com a faixa salarial na qual ele se encontra: 1 → 10%, 2→20%, 3→ 30%, 4→ 40% e 5→ 50%, utilizando o comando FOR.

SQL> ed exemplo9

```
set serveroutput on
```

```
DECLARE
        aux_salario emp.sal%type;
        --
        cursor empregados is
        select  e.ename
                ,e.sal
                ,s.grade
        from    emp e
                ,salgrade s
        where   e.sal between s.losal and s.hisal;
        --
        aux_empregados empregados%rowtype;
BEGIN
        for aux_empregados in empregados loop
        aux_salario := aux_empregados.sal;
        --
        if aux_empregados.grade = 1 then
                aux_salario := aux_salario * 1.1;
        elsif aux_empregados.grade = 2 then
                aux_salario := aux_salario * 1.2;
        elsif aux_empregados.grade = 3 then
                aux_salario := aux_salario * 1.3;
        elsif aux_empregados.grade = 4 then
                aux_salario := aux_salario * 1.4;
        elsif aux_empregados.grade = 5 then
                aux_salario := aux_salario * 1.5;
        end if;
        --
  dbms_output.put_line('Salario de ' ||to_char(aux_salario)
                ||' para '||aux_empregados.ename);
        end loop;
END;
/
Salario de 880 para SMITH
Salario de 1210 para ADAMS
Salario de 1045 para JAMES
Salario de 1500 para WARD
Salario de 1500 para MARTIN
Salario de 1560 para MILLER
Salario de 2080 para ALLEN
Salario de 1950 para TURNER
Salario de 4165 para JONES
```

Salario de 3990 para BLAKE
Salario de 3430 para CLARK
Salario de 4200 para SCOTT
Salario de 4200 para FORD
Salario de 7500 para KING

PL/SQL procedure successfully completed.

CURSORES COM PARÂMETROS

Os cursores podem receber parâmetros de forma que, sempre que forem abertos, retornam resultados diferentes. Para que um cursor possa receber parâmetros, eles devem ser informados no momento da declaração do cursor.

Sintaxe

CURSOR <nome_cursor>(<nome_do_parâmetro1> <tipo_de_dado1>,
<nome_do_parâmetro2> <tipo_de_dado2>,...) IS
<comando SELECT>...

Um cursor que possui parâmetros deve ser aberto de modo que os parâmetros possam ser informados para que a query seja executada.

Sintaxe

OPEN <nome cursor> (<p1>,<p2>,...);

FOR <registro> IN <nome cursor> (<p1>,<p2>,...) LOOP
<comandos>;
END LOOP;

➔ Exemplo
Listar o cadastro de departamentos, mostrando os empregados de cada departamento. Utilizar cursor recebendo parâmetro.

SQL> ed exemplo10

```
set serveroutput on
DECLARE
        aux_deptno dept.deptno%type;
        aux_dname  dept.dname%type;
        --
        cursor deptos is
        select  deptno
                ,dname
        from    dept;
        --
        cursor empregados (prm_deptno number) is
        select  ename
        from    emp
        where   deptno = prm_deptno;
        --
        aux_empregados empregados%rowtype;
BEGIN
        open deptos;
        loop
        fetch deptos
        into  aux_deptno
                ,aux_dname;
        exit when deptos%notfound;
        --
  dbms_output.put_line('Departamento: ' ||aux_dname);
        --
        for aux_empregados in empregados(aux_deptno) loop
  dbms_output.put_line('--> '||aux_empregados.ename);
        end loop;
        end loop;
        close deptos;
END;
/
Departamento: ACCOUNTING
--> CLARK
--> KING
-> MILLER
Departamento: RESEARCH
--> JONES
--> SCOTT
--> ADAMS
```

```
--> FORD
Departamento: SALES
--> ALLEN
--> WARD
--> MARTIN
--> BLAKE
--> TURNER
--> JAMES
Departamento: OPERATIONS

PL/SQL procedure successfully completed.
```

9 UTILIZAÇÃO DE SQL DINÂMICO

A técnica de SQL dinâmico consiste em escrever comandos SQL armazenados em variáveis do alfanuméricas e, posteriormente, submeter o comando à execução. É útil em programas em que dependendo das circunstâncias, executam comandos SQL semelhantes, mas diferentes entre si.

- Package: DBMS_SQL
- Alguns procedimentos e funções da package DBMS_SQL: OPEN_CURSOR,PARSE,BIND_VARIABLE,EXECUTE,FETCH_ROWS,CLOSE_CURSOR, DEFINE_COLUMN, COLUMN_VALUE

➔ Exemplo

```
set serveroutput on
declare
        ORACLE7 constant integer := 2;

        n           integer;
        id_cursor     integer;
        aux_coluna1    number(12,02);
        aux_coluna2    number(12,02);
begin
        id_cursor := dbms_sql.open_cursor;
        -- Busca um número de identificação e abre o cursor

  dbms_sql.parse(id_cursor, 'select ' || '&colunas_da_tabela_pedido' ||
                  ' from pedido where num_pedido = ' ||
                  to_char(&pedido), ORACLE7);
        -- Faz o parse (compilação) do comando
```

```
  dbms_sql.define_column(id_cursor, 1, aux_coluna1);
  dbms_sql.define_column(id_cursor, 2, aux_coluna2);
        -- Define as colunas a serem retornadas

        n := dbms_sql.execute(id_cursor);
                -- Executa o comando com retorno = 0 caso tenho executado OK

        n := dbms_sql.fetch_rows(id_cursor);
                -- Faz o fetch, retornando o número de linhas

  dbms_sql.column_value(id_cursor, 1, aux_coluna1);
  dbms_sql.column_value(id_cursor, 2, aux_coluna2);
        -- retorna o valor do fetch para a variável

  dbms_sql.close_cursor(id_cursor);
        -- fecha o cursor

  dbms_output.put_line('Coluna 1 = ' || aux_coluna1);
  dbms_output.put_line('Coluna 2 = ' || aux_coluna2);
end;
/
Enter value for colunas_da_tabela_pedido: num_pedido,cod_fornec
old  10:   dbms_sql.parse(id_cursor, 'select ' || '&colunas_da_tabela_pedido' ||
new  10:    dbms_sql.parse(id_cursor, 'select ' || 'num_pedido,cod_fornec' ||
Enter value for pedido: 2
old  12:                          to_char(&pedido), ORACLE7);
new  12:                           to_char(2), ORACLE7);
Coluna 1 = 2
Coluna 2 = 1
PL/SQL procedure successfully completed.
SQL>_
```

10 TRATAMENTO DE EXCEÇÕES

Exceção é qualquer problema ocorrido que impossibilita o Banco de Dados de executar determinada instrução no bloco PL/SQL. A instrução pode ser um comando SQL ou uma instrução de atribuição. As exceções podem ser pré-definidas ou definidas pelo usuário.

Quando um erro ocorre, o fluxo é desviado para a seção EXCEPTION do bloco PL/SQL correspondente, onde a exceção deve ser tratada. Se a seção EXCEPTION não existir, a mensagem de erro é devolvida para o usuário. As exceções podem ser disparadas através da instrução RAISE.

Sintaxe:
RAISE <nome da exceção>

No caso de exceções definidas pelo usuário, a exceção deve ser uma variável declarada na seção DECLARE, como sendo do tipo EXCEPTION. Caso seja um erro do tipo ORA, pode ser associado ao erro Oracle através da declaração PRAGMA EXCEPTION_INIT.

Sintaxe:

```
DECLARE
    <nome exceção> EXCEPTION;

    PRAGMA EXCEPTION_INIT (nome exceção, código_erro_oracle);

BEGIN
```

Para especificar qual exceção iniciará a execução do código, deve-se utilizar o comando WHEN. Todas as exceções que não existem um tratamento específico podem ser tratadas através da exceção OTHERS.

Sintaxe:

```
BEGIN
...
EXCEPTION
    WHEN <nome exceção1> THEN
        comandos1
    WHEN <nome exceção2> THEN
        comandos2
    WHEN OTHERS then
        comandos para qualquer outro erro ocorido
END;
```

GERANDO INTERRUPÇÃO DO PROGRAMA

O programa também pode provocar interrupção da execução através do comando *raise_application_error*.

Sintaxe:

RAISE_APPLICATION_ERROR(código_do_erro, mensagem);

Código do erro é o código a ser utilizado para o erro definido pelo usuário. O Oracle reserva os códigos de 20000 até 20999 para esse propósito. O erro será mostrado como qualquer outra mensagem de erro do dos.
EXCEÇÕES PRÉ-DEFINIDAS

. Dup_val_on_index - índice duplicado
Sqlcode = -1 Erro oracle = ORA-00001

. Invalid Number - número inválido na conversão de valores
Sqlcode = -1722 Erro oracle = ORA-01722

. Login_denied - usuário/senha inválidos
Sqlcode = -1017 Erro oracle = ORA-01017

. No_Data_Found - nenhuma linha recuperada
Sqlcode = +100 Erro oracle = ORA-1403

. Not_logged_on - não conectado ao Oracle
Sqlcode = -1012 Erro oracle = ORA-01012

. Program_error - erro interno do PL/SQL.
Sqlcode = -6501 Erro oracle = ORA-06501
. Store_error - problema de memória.
Sqlcode = -6500 Erro oracle = ORA-06500

. Timeout_on_resource - problema de timeout (estouro de tempo no aceso a um objeto da base de dados)
Sqlcode = -51 Erro oracle = ORA-00051

. Too_many_rows - mais de uma linha recuperada
Sqlcode = -1427 Erro oracle = ORA-01427

. Value_error - erro de conversão de dado ou valor atribuído a um campo ou coluna maior que os mesmos.
Sqlcode = -6502 Erro oracle = ORA-06502

```
. Zero_divide          - tentativa de dividir um número por zero.
        Sqlcode = -1476                 Erro oracle = ORA-01476


SQL>ed exemplo11

set serveroutput on

DECLARE
        aux_salario emp.sal%type;
        aux_faixa   salgrade.grade%type;
BEGIN
        select  e.sal
                ,s.grade
        into    aux_salario
                ,aux_faixa
        from    emp e
                ,salgrade s
        where   e.sal between s.losal and s.hisal
        and     e.empno = &prm_empno;
        --
        if aux_faixa = 1 then
                aux_salario := aux_salario * 1.1;
        elsif aux_faixa = 2 then
                aux_salario := aux_salario * 1.2;
        elsif aux_faixa = 3 then
                aux_salario := aux_salario * 1.3;
        elsif aux_faixa = 4 then
                aux_salario := aux_salario * 1.4;
        elsif aux_faixa = 5 then
                aux_salario := aux_salario * 1.5;
        end if;
        --
  dbms_output.put_line('Salario de ' ||to_char(aux_salario));
        --
exception
  when no_data_found then
  dbms_output.put_line('Empregado não cadastrado ou '
                ||' não se enquadra nas faixas pré-cadastradas');
  when others then
        raise_application_error(-20501,'Erro: '||sqlerrm);
END;
/
```

```
Enter value for prm_empno: 111
old  12:    and     e.empno = &prm_empno;
new  12:    and     e.empno = 111;
Empregado não cadastrado ou  não se enquadra nas faixas pré-cadastradas

PL/SQL procedure successfully completed.
```

FONTE: HBTEC. Programando Proceduralmente em PL/SQL . Disponível em: <www.hbtec.com.br>Acesso em: 1º de Abril de 2015.

RESUMO DO TÓPICO 2

Neste capítulo você aprendeu:

- Que PLSQL é uma linguagem imperativa, permitindo que se criem programas inteiros em sua estrutura.
- O que é PL/SQL e conheceu seu ambiente.
- As vantagens da utilização do ambiente PL/SQL e seus tipos de blocos de comandos.
- A executar blocos anônimos de comandos PLSQL.
- A gerar saída a partir de um bloco PL/SQL iSQL*Plus como ambiente de programação PL/SQL.
- A criar códigos PL/SQL para estabelecer interface com o Banco de Dados.
- A desenvolver unidades de programa PL/SQL executadas de forma eficiente.
- A usar estruturas de programação PL/SQL e instruções de controle condicional.
- A tratar erros de runtime e exceções pré-definidas.

AUTOATIVIDADE

Prezado(a) Acadêmico(a)!

As autoatividades que seguem destinam-se à averiguação da aprendizagem deste tópico de estudos. Empenhe-se na sua execução e utilize os serviços disponibilizados pelo NEAD para sanar eventuais dúvidas.

1. Crie a base de dados abaixo:

```
create table BD_EMP
(
  EMP_NUM       NUMBER(5),
  EMP_NOME      VARCHAR2(60),
  DEPTO_NUM     NUMBER(5),
  CARGO_NUM     NUMBER(5),
  EMP_DAT_ADMIS DATE,
  EMP_VAL_SAL   NUMBER(10,2),
  EMP_VAL_COMIS NUMBER(10,2)
);



create table CARGO
(
  CARGO_NUM   NUMBER(5),
  CARGO_NOME  VARCHAR2(60),
  CARGO_SUPER NUMBER(5)
);


create table DEPTO
(
  DEPTO_NUM  NUMBER(5),
  DEPTO_NOME VARCHAR2(60)
);



create table FAIXA_SAL
(
  FAIXA_DESC    VARCHAR2(60),
  FAIXA_VAL_MIN NUMBER(10,2),
  FAIXA_VAL_MAX NUMBER(10,2)
);
```

```
insert into BD_EMP (EMP_NUM, EMP_NOME, DEPTO_NUM, CARGO_NUM,
EMP_DAT_ADMIS, EMP_VAL_SAL, EMP_VAL_COMIS)
values (1, 'SIMONE', 1, 1, to_date('01-09-2010', 'dd-mm-yyyy'), 865, null);
insert into BD_EMP (EMP_NUM, EMP_NOME, DEPTO_NUM, CARGO_NUM,
EMP_DAT_ADMIS, EMP_VAL_SAL, EMP_VAL_COMIS)
values (2, 'DEMIS', 2, 3, to_date('01-06-2010', 'dd-mm-yyyy'), 136500, null);
insert into BD_EMP (EMP_NUM, EMP_NOME, DEPTO_NUM, CARGO_NUM,
EMP_DAT_ADMIS, EMP_VAL_SAL, EMP_VAL_COMIS)
values (4, 'WERNER', 3, 4, to_date('13-09-2010', 'dd-mm-yyyy'), 212700, null);
insert into BD_EMP (EMP_NUM, EMP_NOME, DEPTO_NUM, CARGO_NUM,
EMP_DAT_ADMIS, EMP_VAL_SAL, EMP_VAL_COMIS)
values (5, 'FABIO', 4, 5, to_date('15-09-2010', 'dd-mm-yyyy'), 365400, null);
insert into BD_EMP (EMP_NUM, EMP_NOME, DEPTO_NUM, CARGO_NUM,
EMP_DAT_ADMIS, EMP_VAL_SAL, EMP_VAL_COMIS)
values (6, 'ALEXANDRE', 5, 5, to_date('21-09-2010', 'dd-mm-yyyy'), 458645,
null);
insert into BD_EMP (EMP_NUM, EMP_NOME, DEPTO_NUM, CARGO_NUM,
EMP_DAT_ADMIS, EMP_VAL_SAL, EMP_VAL_COMIS)
values (7, 'JULIA', 4, 3, to_date('13-09-2010', 'dd-mm-yyyy'), 369800, null);
insert into BD_EMP (EMP_NUM, EMP_NOME, DEPTO_NUM, CARGO_NUM,
EMP_DAT_ADMIS, EMP_VAL_SAL, EMP_VAL_COMIS)
values (8, 'ANGELA', 3, 8, to_date('22-09-2010', 'dd-mm-yyyy'), 85600, null);
insert into BD_EMP (EMP_NUM, EMP_NOME, DEPTO_NUM, CARGO_NUM,
EMP_DAT_ADMIS, EMP_VAL_SAL, EMP_VAL_COMIS)
values (9, 'FERNANDA', 2, 10, to_date('01-05-2010', 'dd-mm-yyyy'), 52735, null);
insert into BD_EMP (EMP_NUM, EMP_NOME, DEPTO_NUM, CARGO_NUM,
EMP_DAT_ADMIS, EMP_VAL_SAL, EMP_VAL_COMIS)
values (10, 'ODIRLEI', 6, 6, to_date('17-09-2010', 'dd-mm-yyyy'), 362100, null);
insert into BD_EMP (EMP_NUM, EMP_NOME, DEPTO_NUM, CARGO_NUM,
EMP_DAT_ADMIS, EMP_VAL_SAL, EMP_VAL_COMIS)
values (11, 'RAMON', 7, 7, to_date('23-07-2010', 'dd-mm-yyyy'), 256400, null);
insert into BD_EMP (EMP_NUM, EMP_NOME, DEPTO_NUM, CARGO_NUM,
EMP_DAT_ADMIS, EMP_VAL_SAL, EMP_VAL_COMIS)
values (12, 'RENATA', 8, 8, to_date('14-09-2010', 'dd-mm-yyyy'), 110000, null);
insert into BD_EMP (EMP_NUM, EMP_NOME, DEPTO_NUM, CARGO_NUM,
EMP_DAT_ADMIS, EMP_VAL_SAL, EMP_VAL_COMIS)
values (13, 'FRANCIELLE', 9, 11, to_date('13-09-2010', 'dd-mm-yyyy'), 163500,
null);
commit;
```

```
insert into CARGO (CARGO_NUM, CARGO_NOME, CARGO_SUPER)
values (1, 'GERENTE GERAL', 2);
insert into CARGO (CARGO_NUM, CARGO_NOME, CARGO_SUPER)
values (2, 'DIRETOR', 3);
insert into CARGO (CARGO_NUM, CARGO_NOME, CARGO_SUPER)
values (3, 'PRESIDENTE', null);
insert into CARGO (CARGO_NUM, CARGO_NOME, CARGO_SUPER)
values (4, 'GERENTE FINANCEIRO', 2);
insert into CARGO (CARGO_NUM, CARGO_NOME, CARGO_SUPER)
values (5, 'GERENTE TI', 3);
insert into CARGO (CARGO_NUM, CARGO_NOME, CARGO_SUPER)
values (6, 'PROGRAMADOR', 5);
insert into CARGO (CARGO_NUM, CARGO_NOME, CARGO_SUPER)
values (7, 'ANALISTA SISTEMAS', 5);
insert into CARGO (CARGO_NUM, CARGO_NOME, CARGO_SUPER)
values (8, 'SECRETÁRIA', 7);
insert into CARGO (CARGO_NUM, CARGO_NOME, CARGO_SUPER)
values (10, 'SERVIÇOS GERAIS', 1);
insert into CARGO (CARGO_NUM, CARGO_NOME, CARGO_SUPER)
values (11, 'AUX ADMN', 1);
commit;

insert into DEPTO (DEPTO_NUM, DEPTO_NOME)
values (1, 'RH');
insert into DEPTO (DEPTO_NUM, DEPTO_NOME)
values (2, 'FINANCEIRO');
insert into DEPTO (DEPTO_NUM, DEPTO_NOME)
values (3, 'TI');
insert into DEPTO (DEPTO_NUM, DEPTO_NOME)
values (4, 'EXPORTAÇÃO');
insert into DEPTO (DEPTO_NUM, DEPTO_NOME)
values (5, 'MARKETING');
insert into DEPTO (DEPTO_NUM, DEPTO_NOME)
values (6, 'INFRA ESTRUT');
insert into DEPTO (DEPTO_NUM, DEPTO_NOME)
values (7, 'NOVOS NEGÓCIOS');
insert into DEPTO (DEPTO_NUM, DEPTO_NOME)
values (8, 'NOVOS PRODUTOS');
insert into DEPTO (DEPTO_NUM, DEPTO_NOME)
values (9, 'QUALIDADE');
commit;
```

```
insert into FAIXA_SAL (FAIXA_DESC, FAIXA_VAL_MIN, FAIXA_VAL_MAX)
values ('1', 50000, 99900);
insert into FAIXA_SAL (FAIXA_DESC, FAIXA_VAL_MIN, FAIXA_VAL_MAX)
values ('2', 1000, 160100);
insert into FAIXA_SAL (FAIXA_DESC, FAIXA_VAL_MIN, FAIXA_VAL_MAX)
values ('3', 1602, 250000);
insert into FAIXA_SAL (FAIXA_DESC, FAIXA_VAL_MIN, FAIXA_VAL_MAX)
values ('4', 250100, 310000);
insert into FAIXA_SAL (FAIXA_DESC, FAIXA_VAL_MIN, FAIXA_VAL_MAX)
values ('5', 310100, 408000);
insert into FAIXA_SAL (FAIXA_DESC, FAIXA_VAL_MIN, FAIXA_VAL_MAX)
values ('6', 408100, 501000);
insert into FAIXA_SAL (FAIXA_DESC, FAIXA_VAL_MIN, FAIXA_VAL_MAX)
values ('7', 501100, 600000);
commit;
```

2. Resolva os exercícios abaixo:

a. Fazer um PLSQL com um cursor para listar todos os departamentos. (Exibir o código e nome do departamento).

b. Fazer um PLSQL com um cursor para listar todos os funcionários do departamento de venda.

c. Fazer um PLSQL com um cursor para listar todos os funcionários que possuem vendas e o valor total das vendas de cada um (Exibir o código e nome do funcionário, e o valor total das vendas).

d. Fazer um PLSQL com um cursor para listar todos os funcionários que possuem vendas e o valor total das vendas de cada um (Exibir o código e o nome do funcionário, e o valor total das vendas). Se o total de vendas ultrapassar 150,00, exibir: 'Vendeu bem". Senão, exibir: 'Vendeu Pouco". Exemplo: 01 – Joãozinho – Vendeu bem.

e. Fazer um PLSQL utilizando cursor com parâmetros para listar todos as vendas de cada funcionário (Exibir o código e nome do funcionário, a data da venda e a quantidade total de venda nessa data). OBS.: agrupar por data de venda...*/.

f. Implementar um bloco PLSQL utilizando cursor com parâmetros para listar: nome do cargo, nome do funcionário, nome do produto, quantidade vendida, valor do produto, valor total da venda etc. OBS.: um cursor para o cargo, Um cursor para os funcionários, um cursor para venda/produto.

Ex.: + Presidente – RAMBO - CAMISA SÃO PAULO - 1 - 450,00 = 450,00.

UNIDADE 3

PROGRAMANDO PROCEDURALMENTE EM BD

OBJETIVOS DE APRENDIZAGEM

Ao final desta unidade, você será capaz de:

- entender de que forma os objetos do Banco de Dados interagem entre si para a resolução de problemas;
- preparar um ambiente de desenvolvimento que permita a utilização de uma linguagem de programação em Banco de Dados;
- criar algoritmos e fazer manutenções em objetos do Banco de Dados.
- construir rotinas de programação avançada em Banco de Dados, através dos objetos: procedures, funções, packages e triggers;
- criar, compilar e recompilar os objetos acima especificados.

PLANO DE ESTUDOS

Esta unidade de ensino está dividida em três tópicos, sendo que no final de cada um deles você encontrará atividades que contribuirão para a apropriação dos conteúdos.

TÓPICO 1 – PROCEDEURES E FUNÇÕES

TÓPICO 2 – PACKAGES

TÓPICO 3 – TRIGGERS

PROCEDURES E FUNÇÕES

1 INTRODUÇÃO

Procedures ou funções são blocos PL*SQL armazenados na base de dados para repetidas execuções. São criadas através do SQL*Plus ou SQL*DBA.

Uma procedure nada mais é do um bloco PL/SQL nomeado. A grande vantagem sobre um bloco PL/SQL anônimo é que pode ser compilado e armazenado no Banco de Dados como um objeto de Banco de Dados. Graças a essa característica as procedures são de fácil manutenção, o código é reutilizável e permite que trabalhemos com módulos de programa. Uma procedure é, então, um bloco PL/SQL nomeado que pode aceitar argumentos (também chamado de parâmetros) e pode ser chamada por um programa, uma sessão SQL ou uma trigger.

➔ EXEMPLO

```
Create or Replace procedure P_INSERE is
Begin
Insert into solic_compra
            (num_solic,dat_emiss,dat_previsao,deptno)
      values
     (4,sysdate,sysdate + 15,20);
End;
```

2 SINTAXE PARA A CRIAÇÃO DE UMA PROCEDURE

```
CREATE OR REPLACE PROCEDURE  nome da procedure
                             (argumento1 modo tipo de dado,
                              argumento2  modo tipo de dado,
                              argumentoN modo tipo de dado)

        IS ou AS
              variáveis locais, constantes etc.
        BEGIN
              bloco PL*SQL
        END nome da procedure
```

Onde:

nome da procedure: é o nome da procedure;

argumento ou Parâmetro: é o nome da variável que será enviada ou retornada para a procedure;

modo: é o tipo do argumento: IN (default) - entrada;

tipo de dado: é o tipo de dado do argumento;

bloco PL*SQL: é o corpo da procedure que define a ação executada por ela.

Observações:

- A expressão OR REPLACE é opcional.
- A palavra-chave AS é equivalente a IS, podendo ser usada uma ou outra.
- O bloco PL/SQL deve ser iniciado com BEGIN ou com declaração de variáveis locais se necessário, sem DECLARE.

3 TRANSFERÊNCIA DE VALORES E PASSAGEM DE PARÂMETROS

QUADRO 8 - TIPOS DE PARÂMETROS PARA PROCEDURES E FUNÇÕES

Tipo de variável	Descrição
IN	Passa um valor do ambiente chamador para a procedure.
OUT	Retorna um valor da procedure para o ambiente chamador.
IN OUT	Passa um valor do ambiente chamador para a procedure, e retorna um outro valor da procedure para o ambiente chamador.

FONTE: A autora

➔ EXEMPLO

```
Create or Replace procedure P_AUMENTO
        (prm_empno   in        emp.empno%type,
         prm_sal               out      emp.sal%type) is
    aux_sal emp.sal%type;
        begin
                select sal
               into   aux_sal
                from   emp
                where  empno = prm_empno;
      --
      aux_sal := aux_sal * 1.20;
      --
      prm_sal := aux_sal;
end P_AUMENTO;

        Create or Replace procedure CHAMA_AUMENTO
        is
                aux_empno emp.empno%type;
      aux_sal   emp.sal%type;
        begin

                aux_empno := 7499;
                P_AUMENTO(aux_empno, aux_sal);
                Dbms_output.put_line('Salário com aumento = '
                   ||to_char(aux_sal));

        end CHAMA_AUMENTO;
```

```
SQL> set  serverout  on
SQL> @P_AUMENTO
10  /

Procedure created.

SQL> @Chama_Aumento
13  /

Procedure created.

SQL> set serverout on
SQL> Execute Chama_Aumento

Salário com aumento = 1920

PL/SQL procedure successfully completed.
```

4 SINTAXE PARA A CRIAÇÃO DE UMA FUNCTION

```
CREATE OR REPLACE FUNCTION nome da função
                (argumento1   modo tipo de dado,
                 argumento2  modo tipo de dado,
                 argumentoN modo tipo de dado)

        RETURN tipo de dado  (retorna apenas um parâmetro)

                IS ou AS
                        variáveis locais, constantes etc.
                BEGIN
                        bloco PL/SQL
                END nome da função;
```

Onde:

nome da função: é o nome da função

argumento ou Parâmetro: é o nome da variável que será enviada ou retornada para a função.

modo: é o tipo do argumento (IN, OUT, IN OUT).

tipo de dado: é o tipo de dado do argumento.

RETURN: determina tipo de argumento que será retornado pela função (obrigatório retornar um valor).

bloco PL*SQL: é o corpo da função que define a ação executada por ela.

OBS.: O REPLACE deve ser sempre usado quando a Function já existir e apresentar necessidade de ser alterada.

➔ EXEMPLO

```
Create or Replace function F_AUMENTO
(prm_empno in emp.empno%type)
 return number
is
        aux_sal emp.sal%type;
begin
        select sal
        into   aux_sal
        from   emp
        where  empno = prm_empno;
        --
        return(aux_sal * 1.20);
        --
end F_AUMENTO;
/
```

Para criar a função:

```
@F_AUMENTO;

Function created.
```

Para executar a função:

```
Select F_AUMENTO(7499) from dual;
```

Ou:

```
Declare
        Aux_sal emp.sal%type;
Begin
        Aux_sal := F_AUMENTO(7499);
        Dbms_output.put_line('Salário com aumento = '
                ||TO_CHAR(AUX_SAL));
End;
```

5 DIFERENÇA ENTRE PROCEDURES E FUNÇÕES

Procedure: Pode conter uma lista de argumentos, e pode retornar um ou mais valores.

Função: Pode conter uma lista de argumentos e deve retornar apenas um valor.

6 MANIPULAÇÃO DE EXCEÇÕES EM TEMPO DE EXECUÇÃO

QUADRO 9 - EXCEÇÕES DA EXECUÇÃO DAS ROTINAS

Tipo de exceção	Efeito Apresentado	Método de manipulação
Oracle sem tratamento de exceções	Comunica o erro interativamente.	Omitir a manipulação da exceção da procedure. O Oracle administra.
Definida pelo usuário	Comunica o erro Interativamente ou grava na base de dados.	Chamar a procedure RAISE_APLICATION_ERROR tratando o erro dentro da procedure.
Oracle com tratamento de exceções	Chama uma operação na base de dados ou melhora a mensagem de erro.	Incluir a manipulação da exceção dentro da procedure para cada bloco usando EXCEPTION.

FONTE: A autora

Observação

As exceptions são divididas em duas categorias:

- Número: é a identificação especificada pelo usuário para uma exception que deve estar entre -20000 e -20999.

- Texto:é a mensagem que deve ser retornada.

É possível retornar o erro para o usuário usando:

RAISE_APLICATION_ERROR(número,'texto');

➔ EXEMPLO

```
Create or Replace procedure P_INSERE_SOLIC
(prm_num_solic in solic_compra.num_solic%type,
        prm_dat_emiss in solic_compra.dat_emiss%type,
        prm_dat_previsao in solic_compra.dat_previsao%type,
        prm_deptno in dept.deptno%type)
is
        aux_deptno dept.deptno%type;
begin

        select deptno
        into    aux_deptno
        from   dept
        where  deptno = prm_deptno;

        insert into solic_compra
                (num_solic,
                dat_emiss,
                dat_previsao,
                cod_depto)
        values
                (prm_num_solic,
                prm_dat_emiss,
                prm_dat_previsao,
                aux_deptno);

        commit;

end P_INSERE_SOLIC;

SQL > Execute P_INSERE_SOLIC(5,sysdate, sysdate + 13, 5);

begin P_INSERE_SOLIC(5,sysdate, sysdate + 13, 5); end;

*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at "SCOTT.P_INSERE_SOLIC", line 9
ORA-06512: at line 1
```

➔ EXEMPLO

Usuário tratando a Exceção

Quando tentar remover um produto, verificar se existe.

```
Create or replace procedure P_REMOVE_SOLIC
                (prm_num_solic          in solic_compra.num_solic%type)
        is
        begin
                delete from solic_compra
                        where num_solic = prm_num_solic;

                if sql%notfound then
        raise_application_error(-20200,
        'Solicitação de Compra não Existe');
                end if;
                commit;
        end P_REMOVE_SOLIC;
SQL > Execute P_REMOVE_SOLIC(10);

begin P_REMOVE_SOLIC(10); end;

*
ERROR at line 1:
ORA-20200: Solicitação de Compra não Existe
ORA-06512: at "SCOTT.P_REMOVE_SOLIC", line 8
ORA-06512: at line 1
```

7 GERENCIANDO PROCEDURES E FUNÇÕES

Benefícios de procedures e funções:

- Melhorar a segurança e integridade dos dados.
- Melhorar a performance.
- Alocação de memória.
- Melhorar manutenção/produtividade.

Documentação de procedures e funções

Pode-se obter a documentação e os erros de compilação usando várias visões do dicionário de dados. Pode-se também obter informações em tempo de execução usando-se procedures fornecidas pelo (Debug) do Banco de Dados.

QUADRO 10 - DOCUMENTAÇÃO DAS ROTINAS

Informação Armazenada	Descrição	Método de Acesso
Fonte	**Texto da procedure ou** function	View USER_SOURCE do dicionário de dados
Argumentos	Passagem de parâmetros	Comando DESCRIBE
P-code	Código do Objeto Compilado	Não tem acesso
Erros de Compilação	Erros de Sintaxe PL/SQL	View USER_ERRORS do dicionário de dados ou comando SHOW ERRORS
Informações em tempo de execução	Variáveis e mensagens especificadas	Por procedures do Oracle DBMS_OUTPUT

FONTE: A autora

QUADRO 11 - COMPILAÇÃO DAS ROTINAS

Tarefa	Comando	Circunstância de Erro
Criar uma nova procedure ou função	CREATE PROCEDURE/ FUNCTION	A procedure/ function já existe.
Modificar uma procedure ou function existente.	CREATE OR REPLACE PROCEDURE/FUNCTION	Nenhuma.
Remover uma procedure ou function existente	DROP PROCEDURE/ FUNCTION	Procedure/Function não existe.

FONTE: A autora

8 INVOCANDO PROCEDURES E FUNÇÕES

QUADRO 12 - CHAMADA DAS ROTINAS

Ambiente Chamador	Sintaxe	Argumentos Passados
Bloco PL/SQL	Chamada pelo nome	Variáveis locais
Outra procedure ou função armazenada	Chamada pelo nome	Variáveis locais e globais da package
SQL*Plus	EXECUTE	Parâmetros de substituição SQL*Plus (somente IN); Variáveis globais SQL*Plus (VARIABLE)
SQL*DBA	EXECUTE	
Oracle Forms	Chamada pelo nome	Itens/colunas e variáveis globais do Oracle Forms

Aplicação pré-compilada	EXEC SQL EXECUTE bloco PL*Sql	Variáveis Host (do hospedeiro) e Variáveis locais PL/SQL
Interfaces de chamadas Oracle (OCI)	OSQL3 bloco PL/SQL	Variáveis Host; Variáveis locais PL/SQL

FONTE: A autora

Para invocar uma procedure ou função, passando os argumentos necessários para sua execução, deve-se obedecer às seguintes regras:

➔ **EXEMPLO 1**

Chamada de uma procedure através de algum PL*SQL:

```
Declare
        v_codfornec   number := 10;
begin
        INSERE_FORNEC(v_codfornecedor);
end;
/
```

➔ **EXEMPLO 2**

Chamada de uma procedure através da base de dados:

```
Create or replace procedure PROCESSA_FORNEC
        (v_codfornec in fornecedor.cod_fornec%type)
is
begin
        INSERE_FORNEC(v_codfornec);
end;
/
```

➔ **EXEMPLO 3**

Chamada de uma procedure através do SQL*Plus ou do SQL*dba:

- Invocar usando o comando EXECUTE.

- Usando ACCEPT para aceitar um parâmetro qualquer.

```
SQL> Accept v_cod_fornec prompt 'Entre com o código do Fornecedor: '
Entre com o código do Fornecedor: 2
SQL> Execute INSERE_FORNEC(&v_cod_fornec);
```

9 PASSAGEM DE ARGUMENTOS PARA A PROCEDURE

Existem 3 maneiras de passar argumentos para a procedure:

- Posicional

Passa o valor na mesma ordem como foi definida na procedure.

➔ EXEMPLO

SQL> Execute INSERE_ITEM_PEDIDO(2,4,3,10,50);

- Nominal

Associa o argumento com o valor usando o símbolo =>, não importando a disposição.

➔ EXEMPLO

SQL> Execute INSERE_ITEM_PEDIDO(v_vlr_unit=>50,v_qtd_pedido=>10,-
>v_num_solic=>3, v_num_pedido=>2,v_cod_produto=>4);

- Combinação

Pode passar valores usando o método de posicionamento e nominal, desde que siga um determinado critério.

➔ EXEMPLO

SQL> execute INSERE_ITEM_PEDIDO(2,4,3,v_qtd_pedido=>10,50);

10 INVOCANDO FUNÇÕES

As funções são invocadas da mesma forma como se invoca uma procedure. No entanto, uma função retorna apenas um valor.

QUADRO 13 - RETORNO DE PROCEDURES E FUNÇÕES

Construção	Valor retornado	Local de Uso
Procedure	Valores através do OUT	Substituindo um conjunto de comandos executáveis.
Function	Retorna apenas um valor através do RETURN	Substituindo uma variável ou expressão dentro de um comando.

FONTE: A autora

➔ EXEMPLO

```
CREATE OR REPLACE FUNCTION  consulta_solic_compra
                (v_num_solic  IN   solic_compra.num_solic%type)
                RETURN number       IS
        v_cod_depto   solic_compra.cod_depto%type;
        BEGIN
        SELECT  cod_depto
INTO v_cod_depto
FROM SOLIC_COMPRA
WHERE num_solic = v_num_solic;

        RETURN (v_cod_depto);
        END consulta_solic_compra;
```

11 DEBUG DE PROCEDURES E FUNÇÕES

O Oracle possui procedures DBMS_OUTPUT que possibilitam fazer um debug das procedures e funções em tempo de execução.

QUADRO 14 - DEBUG DAS PROCEDURES E FUNÇÕES

Categoria	**DBMS_OUTPUT Procedure**	**Descrição**
Output	PUT	Inclui textos na procedure, na linha corrente de saída do buffer.
	NEW_LINE	Marca o fim da linha no buffer de saída, apresenta a mensagem e salta para a próxima linha.
	PUT_LINE	Combina a ação de PUT e NEW_LINE.
Input	GET_LINE	Carrega a linha corrente do buffer de saída para dentro da procedure.
	GET_LINES	Carrega um conjunto de linhas do buffer para dentro da procedure.
Outros	ENABLE	Habilita as procedures DBMS_OUTPUT.
	DISABLE	Desabilita as procedures DBMS_OTPUT.

FONTE: A autora

12 USO DE DEBUG DENTRO DE PROCEDURES E FUNÇÕES

Ativando DBMS_OUTPUT através do SQL*Plus ou SQL*DBA com o comando SERVEROUTPUT:

SQL> **SET SERVEROUTPUT ON**;

Preparar o texto para saída dentro da rotina PL*SQL:

DBMS_OUTPUT.PUT(‘texto de saída**’);**

Apresentando display passando para a próxima linha:

DBMS_OUTPUT.NEW_LINE;

➔ EXEMPLO

```
Create or Replace function P_CALCULA
        (v_codproduto          in        item_pedido.cod_produto%type)
        Return number
is
        Cursor cursor_item is
        select num_pedido,
        cod_produto,
        qtd_pedido,
        vlr_unit
        from item_pedido
        where cod_produto = v_codproduto
        order by num_pedido;

        reg cursor_item%rowtype;
        v_totpedidos number;
        v_totgerpedidos number;
        v_cdfornec_wk number;
        v_prim_vez boolean;
        v_nopedido_ant number;

BEGIN
v_prim_vez := true;
v_totgerpedidos := 0;
v_totpedidos := 0;

For reg in cursor_item loop

        if v_prim_vez = true then
        v_nopedido_ant         := reg.num_pedido;
        v_prim_vez     := false;
        End if;

        if reg.num_pedido <> v_nopedido_ant then
        Dbms_output.put('Nro.Pedido: ');
        Dbms_output.put(v_nopedido_ant);
        Dbms_output.put(' Total = ');
        Dbms_output.put(v_totpedidos);
        Dbms_output.new_line;
        v_totpedidos   := 0;
        v_nopedido_ant         := reg.num_pedido;
        End if;

        v_TotPedidos  := v_TotPedidos + reg.qtd_pedido;
        v_totgerPedidos         := v_totgerpedidos + reg.qtd_pedido;
```

```
End Loop;

/* Se houver, Imprime o Ultimo Pedido */

If v_totpedidos <> 0 then
        Dbms_output.put('Nro.Pedido: ');
        Dbms_output.put(v_nopedido_ant);
        Dbms_output.put(' Total = ');
        Dbms_output.put(v_totpedidos);
        Dbms_output.new_line;
End if;

Return(v_totgerpedidos);
Commit;

Exception
        When no_data_found then
                raise_application_error(-20201,'fornecedor invalido');
END;
```

Compilação da Função P_CALCULA

```
SQL> @CalcTot.Sql
Input truncated to 1 characters
 54  /

Function created.
```

Execução da Função P_CALCULA

```
SQL> set verify off
SQL> set serveroutput on
SQL> define v_codproduto = 2
SQL> variable total number
SQL> execute :total := P_CALCULA(&v_codproduto);

Nro.Pedido: 2 Total = 2
Nro.Pedido: 3 Total = 3
Nro.Pedido: 4 Total = 4
Nro.Pedido: 5 Total = 2
Nro.Pedido: 7 Total = 2
Nro.Pedido: 8 Total = 200

PL/SQL procedure successfully completed.
```

```
SQL> Print Total

  TOTAL
----------
       213
```

13 CONTROLE DE SEGURANÇA

Privilégios necessários para executar operações com a procedure ou função:

QUADRO 15 - SEGURANÇA DAS ROTINAS

Operação	**Privilégio Requerido**	**Outros privilégios**
CREATE	Privilégio do System para CREATE PROCEDURE ou CREATE ANY PROCEDURE	Acessar todos os objetos referenciados pela procedure.
CREATE ou REPLACE	Owner da procedure ou privilégio do System para CREATE PROCEDURE ou CREATE ANY PROCEDURE	Idem.
DROP	Owner da procedure ou privilégio do system para DROP ANY PROCEDURE	Não tem outros privilégios.
EXECUTE	Owner da procedure ou privilégio para EXECUTE ou privilégio do system para EXECUTE ANY PROCEDURE	Não tem outros privilégios.

FONTE: A autora

➔ EXEMPLO

Da conta SYSTEM, produzir o privilégio de Seleção e inserção na tabela Depto para o desenvolvedor INFBLU.

```
SQL> GRANT EXECUTE select, insert
2         on Depto to INFBLU;

Grant succeeded.
```

14 DEPENDÊNCIAS PROCEDURAIS

Dependências Diretas e Indiretas

Uma procedure ou função depende **diretamente** de um objeto da base de dados se o objeto estiver referenciado dentro do corpo da procedure, ou seja, se a procedure invocar tal objeto.

QUADRO 16 - DEPENDÊNCIA DE OBJETOS NA BASE DE DADOS

Objeto Dependente	**Objeto Referenciado Diretamente**
Procedure ou função	Tabela
Procedure ou função	View
Procedure ou função	Sequência
Procedure ou função	Procedure ou função

FONTE: A autora

Uma procedure ou função depende **indiretamente** de um objeto da base de dados se ela referenciar um objeto intermediário que depende de outro objeto, causando a ligação de dependências (intermediário).

QUADRO 17 - DEPENDÊNCIA COM OBJETOS INTERMEDIÁRIOS

Objeto Dependente	**Objeto Intermediário**	**Objeto referenciado Indiretamente**
Procedure ou função	View	Tabela
Procedure ou função	View	View
Procedure ou função	Procedure ou Função	(Ver tabela anterior)

FONTE: A autora

15 DEPENDÊNCIAS LOCAIS E REMOTAS

QUADRO 18 - TIPOS DE DEPENDÊNCIA

Tipo de dependência	Descrição	Significado
Local	Objetos do mesmo nó	O Oracle trilha as dependências e recompila os objetos dependentes automaticamente.
Remota	Objetos em nós separados	O Oracle não trilha as dependências e não contacta os objetos dependentes.

FONTE: A autora

Quando a dependência for local, ao alterar a tabela os objetos que referenciam esta tabela, direta ou indiretamente, ficarão com *status* INVALID. Quando a dependência for remota, ao alterar uma tabela, os objetos que referenciam a tabela, direta ou indiretamente, ficarão com *status* INVALID, exceto os objetos que estejam ligados remotamente, pois estes objetos pertencem a outro dicionário de dados.

Para visualizar qual objeto que é chamado por outro no dicionário de dados, utilizar a view USER_DEPENDENCIES. Para exibir as dependências indiretas de outros objetos, usamos as tabelas DEPTREE e IDEPTREE. Para criar estas tabelas, usar o script utldtree.sql. Populam-se estas tabelas com informações para um objeto referenciado, invocando a procedure DEPTREE_FILL.

Compilando dependências locais explicitamente

Sintaxe para recompilar procedures:

ALTER PROCEDURE nome procedure COMPILE

Sintaxe para recompilar funções:

ALTER FUNCTION nome função COMPILE

Casos de alterações e compilações

QUADRO 19 - RETORNO DA COMPILAÇÃO DOS OBJETOS

Objeto Referenciado	**Exemplo Alteração**	**Compilação Procedure**
Tabela	Tabela é excluída ou renomeada	Malsucedida
Tabela	Adição de coluna	Bem-sucedida se não usar %ROWTYPE
Tabela	Troca do tipo de dado de uma coluna referenciada	Malsucedida se não usar %TYPE ou %ROWTYPE
Tabela	Troca do tipo de dado de uma coluna não referenciada	Bem-sucedida
View	A view é regravada com colunas diferentes	Malsucedida se não usar %ROWTYPE
View	A view é regravada com as mesmas colunas	Bem-sucedida
Sequência	Sequência é excluída	Malsucedida
Procedure	Lista de parâmetros é alterada	Malsucedida
Procedure	Corpo PL/SQL é modificado	Bem-sucedida

FONTE: A autora

- Para verificar o sucesso da compilação, utilizar a view USER_OBJECTS.

➔ EXEMPLO

```
SQL> select object_name        ,object_type     ,status
2   from user_objects
3   where object_type = 'PROCEDURE';

OBJECT_NAME                    OBJECT_TYPE                    STATUS
-----------------    --------------------  -----------
P_INSERE_PRODUTO          PROCEDURE                INVALID
P_REMOVE_PRODUTO        PROCEDURE              VALID
```

- A mudança da segurança não invalida os objetos dependentes, mas pode causar erros na execução deles.

QUADRO 20 - ERROS NA EXECUÇÃO DAS ROTINAS APÓS COMPILAÇÃO

Objeto Referenciado	Exemplo de Alteração	Resultado em tempo de execução
Tabela	Tirar privilégios de objetos que interferem	Malsucedido
Tabela	Tirar privilégios de objetos que não interferem	Bem-sucedido

FONTE: A autora

16 MECANISMO AUTOMÁTICO DE RECOMPILAÇÃO

O Banco de Dados recompila os objetos com *status* INVALID automaticamente quando eles são invocados, antes de serem executados.

17 MECANISMO AUTOMÁTICO DE DEPENDÊNCIAS LOCAIS

- O Oracle trilha dentro do dicionário de dados as dependências entre todos os objetos da base de dados.

- Registra um *status* para cada objeto (VALID ou INVALID).

- Ao alterar qualquer objeto da base, marca como inválido todos os objetos dependentes daquele que foi alterado.

- Quando uma procedure inválida é invocada para executar, o Oracle recompila a procedure primeiro antes de executá-la.

- Dentro de uma chamada de dependências, o Oracle valida os objetos da base automaticamente.

18 MECANISMO AUTOMÁTICO DE DEPENDÊNCIA REMOTA

- As procedures que estão na mesma base de dados da tabela alterada (local) têm seu *status* alterado para INVALID.

- As procedures que não estão na mesma base da tabela alterada (remota permanecem com o *status* inalterado.

QUADRO 21 - REFERÊNCIAS PROCEDURAIS X TIPOS DE COMPILAÇÃO

Referência Remota	Recompilação Manual	Recompilação Automática
Procedure ou Função	Disponível	Disponível
Tabela, view ou sequência	Disponível	Não Disponível

FONTE: A autora

19 MECANISMO AUTOMÁTICO DE DEPENDÊNCIA REMOTA (TIMESTAMP)

1. O Oracle registra o timestamp de todas as procedures em tempo de compilação.

a) Quando a procedure remota compilar, o Oracle registra o timestamp dentro da procedure remota.

b) Quando uma procedure local é compilada, o Oracle registra o timestamp da sua compilação, e registra o timestamp da procedure remota na procedure local.

2. Quando a procedure local é invocada, o Oracle compara o timestamp da procedure local com a remota referenciada.

a) Se os timestamps são iguais, indicando que a procedure remota não tem recompilação mais recente que a procedure local, o Oracle executará as procedures sem recompilá-las.

b) Se os timestamps são diferentes, indicando que a procedure remota tem uma recompilação mais recente que a procedure local, o Oracle invalida a procedure local e retorna um erro de execução.

3. Se a procedure local que está agora como INVALID, for invocada uma segunda vez, o Oracle recompilará antes de executá-la, de acordo com o mecanismo automático de dependência local.

a) Assumindo que não existiram erros de compilação, o Oracle guarda o novo timestamp da procedure remota dentro do P code da procedure local, e executará a procedure local sem problemas.

b) Se existirem erros de compilação da procedure local, o status continuará inválido, e o Oracle retornará erros em tempo de execução.

LEITURA COMPLEMENTAR

CIENTISTA DE DADOS: A PROFISSÃO DA MODA

Por Marcos Paulo Faria Lima Barreto

Antes da revolução da internet banda larga e das redes sociais, computadores eram de domínio apenas de quem trabalhava com eles, como engenheiros e profissionais de Tecnologia da Informação. Nos últimos anos, a tecnologia se pulverizou e, segundo o relatório de indicadores da ANATEL, desde 2007 a quantidade de pontos de acesso à rede por banda larga cresceu de 7,7 milhões para perto de 21 milhões no Brasil. Além disso, o avanço das tecnologias de hardware, a redução dos custos de acesso e a criação de aplicativos gratuitos inseriram muitas pessoas no cenário digital.

Com inúmeros dados e informações sobre clientes, mercados e empresas, executivos e profissionais atrelados a tomadas de decisão passaram a enxergar novas possibilidades de fazer negócio e de alavancá-lo. A integração de informações vindas de diversas fontes, como redes sociais e bancos de dados internos das companhias, traz a possibilidade de entender melhor os consumidores e fornecer produtos mais próximos aos seus desejos e necessidades.

Nesse contexto, as pessoas têm um papel importantíssimo neste processo. Elas são responsáveis por explorar os dados, desenvolver os modelos matemáticos que melhor atendem às necessidades do negócio, além de vislumbrar novas oportunidades que possam gerar diferenciais baseadas nos dados.

O cientista de dados, nome dado ao profissional desta área, vive em três mundos: o dos negócios, o da matemática e o de TI. Sua função é transformar os dados disponíveis em balizadores de decisões a serem tomadas. Esse processo de trabalho com dados exige que este profissional tenha qualificações na área de TI para que consiga acessar e processar o dado de forma eficiente e em tempo hábil, capacidades matemáticas para entender as implicações dos modelos utilizados e de negócio para que possa traduzir tudo isso em relatórios que possibilitem decisões assertivas.

Profissionais que transitem entre esses campos não são comuns no mercado, uma vez que o processo de formação profissional foi durante muito tempo segregador dos conhecimentos por áreas. Assim, hoje encontramos profissionais com formação em TI, matemática, estatística, análise de negócios ou engenharia e que fizeram especializações complementares para trabalharem com dados. Apesar disso, é possível visualizar no LinkedIn que muitas das vagas para "data scientist" requerem um "full stack engineer", alguém que domina todo o processo de ciência de dados.

Para solucionar o problema acadêmico, as empresas segmentam o processo de análise de dados e contratam profissionais com conhecimento em, pelo menos, duas áreas complementares.

Alguns cursos novos vêm chamando atenção por tentarem preencher esta lacuna de formação. Os cursos de matemática aplicada de universidades como UFRJ, PUC-Rio e FGV acrescentaram matérias da área de TI e negócios, possibilitando aos seus alunos a entrada no mercado de dados.

Há também uma iniciativa recente do site coursera.com que traz um curso on-line de especialização em ciência de dados, que é certificado pela Universidade de Johns Hopkins, 15ª no ranking da The World University Rankings. Vale ressaltar que, apesar de existirem muitos cursos de tecnólogos em processamento de dados, o tipo de formação oferecida por estes, normalmente, não dá a qualificação necessária para o profissional atuar como cientista de dados.

Entenda essa nova profissão

Com as oportunidades de explorar o Big Data aumentando, executivos voltam-se para um desafio igualmente crescente: achar profissionais especializados e cientistas de dados capazes de capturar, armazenar, gerenciar e analisar grandes volumes de dados que devem ser interpretados e utilizados de maneira coerente e concisa de modo que a empresa possa utilizá-los a seu favor, entendendo quem são seus clientes, como eles se comportam e como eles se relacionam com o produto ou serviço oferecido.

Surge, então, a profissão de Cientista de Dados, considerada a profissão do futuro.

O profissional dessa área vive em três mundos: o dos negócios, o da matemática e o de TI. Ele deve transformar os dados disponíveis em balizadores de decisões a serem tomadas. Esse processo de trabalho com dados exige que este profissional tenha qualificações na área de TI para que consiga acessar e processar o dado de forma eficiente e em tempo hábil, capacidades matemáticas para entender as implicações dos modelos utilizados e de negócio para que possa traduzir tudo isso em relatórios que possibilitem decisões assertivas.

Habilidades necessárias

O cientista de dados tem que saber programação, ser capaz de criar modelos estatísticos e ter o conhecimento e domínio apropriado de negócios.

Precisa também compreender as diferentes plataformas de Big Data e como elas funcionam. Usualmente, esse profissional é formado em estatística, matemática ou ciências da computação. "Peça-chave na estratégia de adoção do Big Data, o cientista de dados vai fazer a ponte entre o gestor de Tecnologia e o Gestor Financeiro, que é quem está à frente dos investimentos em Big Data. Ele sabe a importância do dado para o negócio da empresa", sustenta Dênis Arcieri, presidente da IDC Brasil.

Demanda do mercado

A demanda de cientistas de dados chegará a 4,4 milhões em 2015, com a América Latina, respondendo por quase 1 milhão desses especialistas. Porém, profissionais que transitem entre esses campos não são comuns no mercado, uma vez que o processo de formação profissional foi durante muito tempo segregador dos conhecimentos por áreas. Assim, hoje, encontramos profissionais com formação em TI, matemática, estatística, análise de negócios ou engenharia e que fizeram especializações complementares para trabalharem com dados.

O cientista de dados não é um especialista de fácil disponibilidade. Ao contrário. Empresas começam a se mobilizar para treinar esses profissionais. Estudo recente divulgado pela EMC - que já está capacitando profissionais brasileiros para o Big Data - revela que a maior barreira para lidar com ele, de acordo com 73% das empresas entrevistadas pelo levantamento, é a própria cultura.

FONTE: Disponível em: <http://corporate.canaltech.com.br/noticia/carreira/Cientista-de-Dados--A-profissao-da-moda/#ixzz3VOmaxHl1> Acesso em: 25 mar. 2015.

RESUMO DO TÓPICO 1

Neste tópico, você aprendeu:

- A criar algoritmos em forma de objetos do Banco de Dados, através de procedures e funções.
- A criar, editar, recompilar e fazer manutenções em procedures e funções.
- Que existe uma dependência direta entre os objetos da base de dados.
- Que procedures servem para qualquer propósito, enquanto as funções desempenham um objetivo bem específico.

AUTOATIVIDADE

1. Explique a diferença entre procedures e funções.

2. Diferencie os parâmetros IN e OUT usados em procedures e funções.

3. Crie as tabelas abaixo:

Considere a tabela de empregado (EMP) criada no tópico 1 para a solução do exercício proposto: crie uma procedure e uma função para calcular o reajuste de salário de um funcionário passado como parâmetro, de acordo com um percentual de reajuste também passado como parâmetro.

PACKAGES

1 INTRODUÇÃO

A package é dividida em duas partes: a Especificação e o Corpo da Package. Dentro da especificação declaram-se as construções **públicas**, e dentro do corpo declaram-se somente as construções **privadas**.

QUADRO 22 - DEPENDÊNCIA DE OBJETOS NA BASE DE DADOS

Escopo da construção	**Descrição**	**Local dentro da package**
Público	Disponível para procedure e funções mesmo fora da package	Declarado dentro da especificação da package e definido dentro do corpo da package
Privado	Disponível somente para aquela package	Declarado e definido dentro do corpo da package

FONTE: A autora

Armazenar procedimentos e funções em pacotes permite obter inúmeras vantagens, como encapsulamento e sobrecarga de funções e procedures.

Leia mais em: Stored Procedures, Functions e Packages em Bancos de Dados Oracle. Disponível em: <http://www.devmedia.com.br/stored-procedures-functions-e-packages-em-bancos-de-dados-oracle/25390#ixzz3SP2mtzU7>.

2 PASSOS PARA DESENVOLVER UMA PACKAGE

1. Escrever o texto do comando CREATE PACKAGE dentro de um arquivo texto para criar a especificação da package.

2. Escrever o texto do comando CREATE PACKAGE BODY dentro de um arquivo texto para criar o corpo da package.

3. Executar os dois comandos, com a fonte compilado e armazenar os dois comandos dentro da base de dados.

4. Invocar qualquer construção pública dentro da package de um ambiente Oracle.

3 DECLARAÇÃO DE CONSTRUÇÕES PÚBLICAS NA ESPECIFICAÇÃO DA PACKAGE

Sintaxe:
CREATE OR REPLACE PACKAGE nome da package
IS ou AS
variáveis/constantes
cursores
exceções
procedures
funções
END nome da package;

➔ EXEMPLO

```
CREATE OR REPLACE PACKAGE SALARIO_Package IS
g_per  number := 0.1;
Procedure P_Aumento(prm_empno in number
                    ,prm_sal       out number);
Function F_Desconto(prm_empno in number) return number;
end SALARIO_Package;
/

create or replace PACKAGE BODY Salario_Package is

Function F_Desconto (prm_empno in        number)return number
is
aux_sal number;
BEGIN
```

```
Select sal
Into   aux_sal
from   emp
where  empno = prm_empno;

aux_sal  := aux_sal * g_per;

return(aux_sal);

END F_Desconto;

Procedure P_Aumento (prm_empno in number
                ,prm_sal out number)
is
aux_sal number;
Begin

Select sal
Into   aux_sal
from   emp
where  empno = prm_empno;

prm_sal  := aux_sal * g_per;

End P_Aumento;

End Salario_Package;
/


SQL> @Salario.Sql

Package created.

Package body created.

SQL> declare
  2    aux_sal number;
  3  begin
  4    salario_package.p_aumento(7499,aux_sal);
  5    dbms_output.put_line('Salario com aumento = '
  6             ||to_char(aux_sal));
  7  end;
  8  /
```

```
Salario com aumento = 160

PL/SQL procedure successfully completed.

SQL> declare
  2    aux_sal number;
  3  begin
  4    aux_sal := salario_package.f_desconto(7499);
  5    dbms_output.put_line('Salario com aumento = '
  6                    ||to_char(aux_sal));
  7  end;
  8  /

Salario com aumento = 160

PL/SQL procedure successfully completed.
```

4 CONSTRUÇÕES PÚBLICAS

➔ EXEMPLO

Declaração de uma variável pública para armazenar um valor que pode ser referenciado ou alterado fora da package; declarar uma procedure ou função pública para implementar uma rotina que será invocada repetidamente fora da package.

Criação de uma package que irá calcular o valor total de pedidos, determinando o percentual de 10% de comissão para o rateio entre o pessoal do departamento de Vendas:

```
CREATE OR REPLACE PACKAGE TotPedidos_package IS

g_per_comis   number := 0.1;      /* Iniciada com 10%, var pública */
PROCEDURE INICIALIZA_SOMA_PEDIDOS (v_comissao in number);
                                  /* procedure pública */

        END TotPedidos_package;
```

5 DEFINIÇÃO DE CONSTRUÇÕES PÚBLICAS E PRIVADAS DENTRO DO CORPO DA PACKAGE

Sintaxe:

CREATE OR REPLACE PACKAGE BODY nome da package
IS ou AS
    variáveis
    cursores
    exceções
    procedures
    funções
END nome da package;

➔ EXEMPLO

```
create OR REPLACE PACKAGE Soma_Pedidos_Package IS
g_per_comis Number := 0.1;
Procedure Inicializa_Comissao(v_comissao in number);
Function F_Total_Pedidos(v_comissao in number) return number;
end Soma_Pedidos_Package;
/

create or replace PACKAGE BODY Soma_Pedidos_Package is
Function F_Total_Pedidos
(v_comissao  in        number)
return number
is

Cursor cursor_item is
Select num_pedido,
 cod_produto,
 qtd_pedido,
 vlr_unit
from item_pedido
order by num_pedido;

reg cursor_item%rowtype;
v_totpedidos number;

BEGIN

v_totpedidos := 0;
For reg in cursor_item loop
v_totPedidos := v_totpedidos + reg.qtd_pedido;
End Loop;
```

```
v_TotPedidos    := v_TotPedidos * v_comissao;
return(v_totpedidos);
commit;
END f_Total_Pedidos;

Procedure Inicializa_Comissao
(v_comissao in number)
is
v_totComis number;

Begin

v_Totcomis := 0;
v_TotComis := F_Total_Pedidos(v_comissao);
dbms_output.put_line('Valor Total da Comissao: ' || v_totcomis);
dbms_output.new_line;
if v_TotComis < 100 then
Dbms_output.put_line('Faturamento nao atingiu a meta');
end if;

End Inicializa_Comissao;
   End Soma_Pedidos_Package;
/


SQL> @SomaCom.Sql

Package created.

Package body created.

SQL>  execute soma_pedidos_package.inicializa_comissao(.15);

Valor Total da Comissao: 60.9
Faturamento nao atingiu a meta

PL/SQL procedure successfully completed.
```

6 DOCUMENTAÇÃO DE PACKAGES

QUADRO 23 - DOCUMENTAÇÃO DE PACKAGES

Tarefa	Comando
Criar uma nova especificação da package	CREATE PACKAGE
Criar um novo corpo da package	CREATE PACKAGE BODY
Alterar a especificação de uma package existente	CREATE OR REPLACE PACKAGE
Alterar o corpo de uma package body existente	CREATE OR REPLACE PACKAGE BODY
Remover a especificação e o corpo da package	DROP PACKAGE
Remover somente o corpo da package	DROP PACKAGE BODY

FONTE: A autora

Observações:

- É necessário alterar o corpo da package quando for alterada a especificação da package;

- Não é necessário alterar ou excluir a especificação da package quando for alterado ou excluído o corpo da package.

7 INVOCANDO CONSTRUÇÕES DAS PACKAGES

Depois de armazenada na base de dados, pode-se invocar uma construção da package de dentro da package ou de fora dela, se a construção for privada ou pública respectivamente.

Procedures e Functions

Quando invocar uma procedure ou função da package de dentro dela, não é necessário estar precedido pelo nome da package.

➔ EXEMPLO

Invocando uma função de uma package por uma procedure da package.

```
create or replace PACKAGE BODY Soma_Pedidos_Package is

<demais procedimentos>

Procedure Inicializa_Comissao
(v_comissao in number)
is
v_totComis number;

Begin

v_Totcomis := 0;

V_TOTCOMIS := F_TOTAL_PEDIDOS(V_COMISSAO);

dbms_output.put_line('Valor Total da Comissao: ' || v_totcomis);
dbms_output.new_line;
if v_TotComis < 100 then
Dbms_output.put_line('Faturamento nao atingiu a meta');
end if;

End Inicializa_Comissao;

End Soma_Pedidos_Package;
```

Quando uma procedure ou função é invocada de fora da package, deve-se usar na chamada o nome da procedure precedido pelo nome da package.

➔ EXEMPLO 1

Invocando uma procedure a partir do SQL*Plus. Chamar a procedure INICIALIZA_COMISSAO do SQL*Plus, passando como parâmetro para cálculo do faturamento mínimo o percentual de 15%.

```
SQL> execute soma_pedidos_package.inicializa_comissao(.15);
```

➔ EXEMPLO 2

Invocando uma procedure de uma package em uma outra conta Oracle. Chamar a procedure INICIALIZA_COMISSAO que está localizada na conta INFBLU do SQL*Plus, passando como parâmetro para cálculo do faturamento mínimo o percentual de 15%.

```
SQL> execute infblu.soma_pedidos_package.inicializa_comissao(.15);
```

➔ EXEMPLO 3

Invocando uma procedure de uma package em uma base de dados remota. Chamar a procedure INICIALIZA_COMISSAO que está localizada na conta INFBLU na base de dados determinada pelo database link com o nome de *nk* a partir do SQL*Plus, passando como parâmetro para cálculo do faturamento mínimo o percentual de 15%.

```
SQL> execute infblu.soma_pedidos_package.inicializa_comissao@nk (.15);
```

Variável global - Pública

Da mesma forma, quando referenciar uma variável, cursor, constante ou exceção de dentro de uma package, não é necessário chamar usando o nome da package. Quando referenciar uma variável, cursor, ou exceção de fora da package, é necessário realizar a chamada do objeto precedido pelo nome da package.v

➔ EXEMPLO

Calcula o Valor unitário do produto de código 2, utilizando o percentual da variável global de nome g_per_comis, contida na package de nome Soma_ Pedidos_Package.

```
create or replace procedure CALCULA_VLR_PRODUTO
        (v_num_pedido          in pedido.num_pedido%type,
        v_cod_produto          in produto.dsc_produto%type,
        v_num_solic    in item_solicitacao.num_solic%type,
        v_qtd_pedido           in item_pedido.qtd_pedido%type)

IS
Cursor cursor_item is
Select  num_pedido,
cod_produto,
qtd_pedido,
vlr_unit
from item_pedido
where cod_produto = 2
order by num_pedido;

reg cursor_item%rowtype;
v_vlr_unit      item_pedido.vlr_unit%type;
```

```
Begin
if v_cod_produto = 2 then
v_vlr_unit := (v_vlr_unit * SOMA_PEDIDOS_PACKAGE.G_PER_COMIS) + v_
vlr_unit;
End if;
For reg in cursor_item loop
reg.vlr_unit := v_vlr_unit;
End Loop;
end CALCULA_VLR_PRODUTO;

SQL> @Calcula.Sql
26  /
Procedure created.
```

Quando carregada pela primeira vez a situação de uma variável ou cursor de uma package, persiste por toda a sessão, desde a primeira referência da variável ou cursor até o usuário desconectar-se.

Controlando a situação da variável dentro de uma package:

1. Inicializa a variável no local da sua declaração ou dentro de um procedimento automático somente uma vez (BEGIN).

2. Altera o valor da variável por meio de procedures ou functions na package.
3. A variável pode ser usada por qualquer ferramenta dentro da sessão.

4. Perde o valor da variável após o usuário desconectar-se.

Observações

A situação de uma variável ou cursor persiste através de transações dentro de uma sessão.

A situação não persiste de sessão para sessão para o mesmo usuário.

A situação não persiste de um usuário para outro usuário.

QUADRO 24 - CONTROLE DE SEGURANÇA

Tarefa	Package	Procedure Stand-Alone
Verificar e Documentar	Examinar as visões USER_OBJECTS e USER_SOURCE do dicionário de dados	Examinar as visões USER_OBJECTS e USER_SOURCE do dicionário de dados
Detectar erros de compilação	Examinar a visão USER_ERRORS	Examinar a visão USER_ ERRORS
Facilidade de desenvolvimento	Produzir arquivos de lote SQL*Plus	Produzir arquivos de lote SQL*Plus
Controlar a segurança para o desenvolvedor	Obter o privilégio CREATE PROCEDURE	Obter o privilégio CREATE PROCEDURE
Controlar a segurança para o usuário	Obter o privilégio EXECUTE para a package	Obter o privilégio EXECUTE para a procedure

FONTE: A autora

8 MECANISMO AUTOMÁTICO DE DEPENDÊNCIA

Como acontece com procedures stand-alone, recompila-se packages inválidas manualmente e através do mecanismo automático de dependência local e remoto.

QUADRO 25 - DEPENDÊNCIA DE PACKAGES

Tarefa	Package	Procedure Stand-Alone
Identificar dependências	Examine a view USER_DEPENDENCIES do dicionário de dados	Examine a view USER_DEPENDENCIES do dicionário de dados
Recompilar manualmente	Utilizar o comando ALTER PACKAGE	Utilizar o comando ALTER PROCEDURE
Recompilar automaticamente	Disparo do mecanismo de dependências, local e remoto	Disparo do mecanismo de dependências, local e remoto

FONTE: A autora

Para recompilar Especificação e Corpo da Package:

ALTER PACKAGE nome da package COMPILE PACKAGE

Para recompilar somente a Especificação da Package:

ALTER PACKAGE nome da package COMPILE PACKAGE SPECIFICATION

Para recompilar somente o Corpo da Package:

ALTER PACKAGE nome da package COMPILE PACKAGE BODY

ROLES: *Podem consistir em object e system privileges.
* Ninguém é dono de nenhum esquema.
*Pode ser dado para qualquer Role ou usuário.
*Pode ser habilitado ou desabilitado para cada usuário.
*Pode requerer uma senha especial.

OBS.: As descrições dos Roles são armazenadas no dicionário de dados.

9 PACKAGES PRÉ-DEFINIDAS

QUADRO 26 - PACKAGES DO BANCO DE DADOS

Package Pré - definida	Funcionalidade
DBMS_OUTPUT	Display de informações através de procedures agrupadas (debug)
DBMS_DDL	Compilar procedures, funções e packages obtendo estatísticas com ANALYZE. Usada para compilação de procedures dependentes.
DBMS_SESSION	Fornece informações sobre a sessão SQL.
DBMS_TRANSACTION	Controlar transações e melhorar a performance de pequenas transações locais.
DBMS_PIPE	Enviar mensagens da base de dados para uma aplicação e entre aplicações.
DBMS_ALERT	Avisar que ocorreu um evento na base de dados.
DBMS_LOCK	Requisita, converte e libera bloqueios do usuário que são gerenciados pelos serviços de gerenciamento de bloqueios RDBMS.
DBMS_JOB	Permite que se programe a execução de procedimentos automaticamente.
UTL_FILE	Adiciona recursos de entrada e saída ao PL/SQL.

FONTE: A autora

10 BENEFÍCIOS DO USO DE PACKAGES

- Melhorar a Organização de procedures e funções.
- Melhorar o gerenciamento de procedures e funções armazenadas.
- Melhorar a segurança de procedures e funções armazenadas.
- Inicialização de identificadores para a sessão usuária.
- Melhorar a performance.

11 OUTROS CONCEITOS DE PACKAGES

Para que o Oracle Server execute uma instrução SQL que chame uma "function", ele deve ter certeza de que a função esteja sem "efeitos colaterais". Efeitos colaterais são alterações em tabelas de Bancos de Dados ou variáveis públicas empacotadas (declaradas em uma especificação de pacote). As seguintes restrições aplicam-se às funções armazenadas chamadas a partir de expressões SQL:

- Uma função não pode modificar tabelas, portanto, ela não pode executar INSERT, DELETE, UPDATE ou DELETE.
- Somente funções locais podem atualizar as variáveis do pacote.
- Funções remotas não podem ler ou gravar variáveis de pacotes remotos.
- Funções que leem ou gravam variáveis de pacote não podem usar a opção de consulta paralela.
- Chamadas para subprogramas que quebrem as restrições acima não são permitidas.

RESUMO DO TÓPICO 2

Neste tópico, você aprendeu:

- Que uma package é uma coleção de objetos do Banco de Dados, como procedures, funções, variáveis, constantes e cursores.
- Que a package contém subprogramas que podem ser chamados pelas triggers, procedures ou funções.
- Que a package é uma grande aliada do desenvolvedor, pois permite organizar melhor o código de programação dos sistemas, em módulos.

AUTOATIVIDADE

1. Crie um Package que insere uma nova equipe e logo em seguida imprime a tabela de equipes atualizada

TRIGGERS

1 INTRODUÇÃO

Conceito de Triggers: é um bloco PL/SQL executado de forma implícita sempre que ocorre um determinado evento. Pode ser tanto um gatilho de banco de dados ou um gatilho de aplicação.

➔ EXEMPLO

A cada inclusão na tabela Solic_compra, a Trigger deverá emitir uma mensagem ao usuário, caso a data de previsão for menor que a data de emissão da solicitação de compra.

```
Create or replace trigger Tr_Solic_compra
before insert on solic_compra
for each row
declare
e_previsao_invalida  exception;
begin
If :new.dat_previsao < :new.dat_emiss then
raise e_previsao_invalida;
End if;
Exception
  when e_previsao_invalida then
    raise_application_error(-20100, 'Numero do Pedido: '||
      :new.num_solic || ' com a Data Prevista Invalida');
End;

SQL> @VerPrev.Sql
 20  /
          Trigger created.
```

```
SQL> insert into solic_compra
2       (num_solic
3       ,cod_depto
4       ,dat_emiss
5       ,dat_previsao)
6    values(8,2,'15-jun-97','14-jun-97');
,cod_depto
      *
ERROR at line 3:
ORA-20100: Numero do Pedido: 8 com a Data Prevista Invalida
ORA-06512: at "SCOTT.TR_SOLIC_COMPRA", line 9
ORA-04088: error during execution of trigger 'SCOTT.TR_SOLIC_COMPRA'.
```

Observações

- A declaração <FOR EACH ROW>, e a referência < :NEW > são utilizados para verificar cada linha de inserção na tabela referenciada pela trigger, causando ou não alguma exceção.

- O nome do campo após a declaração <:new> refere-se às colunas da tabela que estão recebendo a operação sobre os dados.

As Triggers são usadas para realizar tarefas relacionadas com validações, restrições de acesso, rotinas de segurança e consistência de dados. Desta forma estes controles deixam de ser executados pela aplicação e passam a ser executados pelos Triggers em determinadas situações:

- Quando temos que popular campos de tabelas que estão desatualizados.
- Em uma migração, com certeza envolverá uma Trigger que ajude a manter a integridade dos dados.

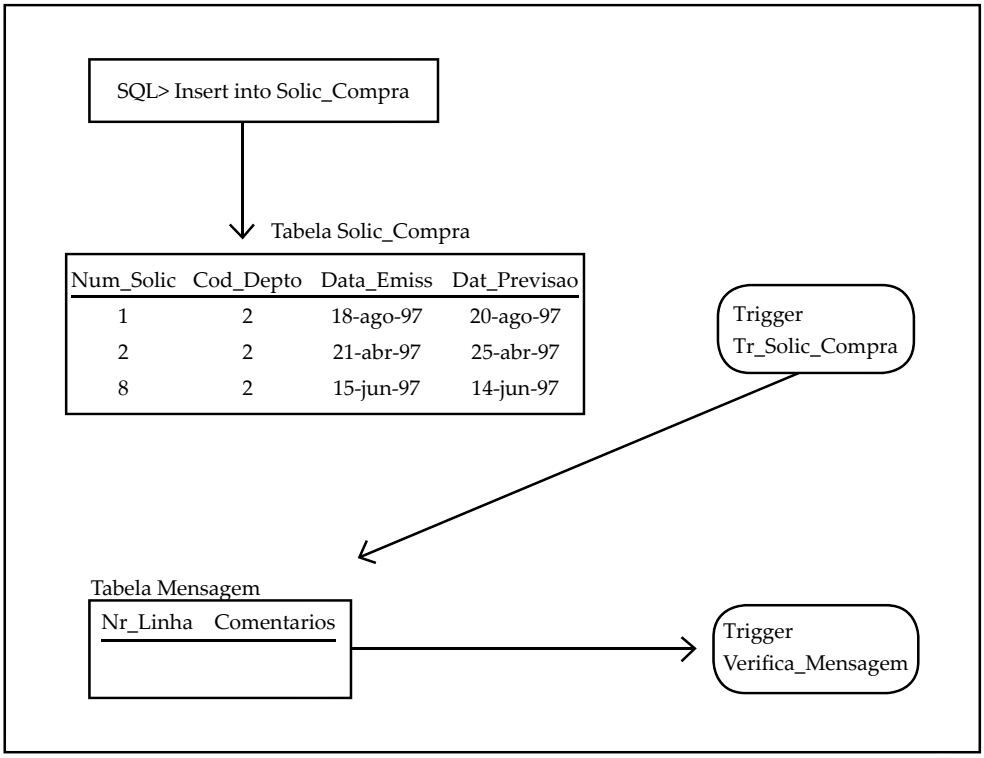
2 EFEITO CASCATA EM DATABASE TRIGGERS

Algumas vezes a ação de um Trigger aponta para outro Trigger, causando o disparo de um segundo Trigger. O número de Triggers em cascata é limitado de acordo com o parâmetro MAX_OPEN_CURSORS, que é setado de acordo com a plataforma de trabalho. Por default o número de Triggers usados ao mesmo tempo é 32.

➔ EXEMPLO

SQL> **insert into solic_compra …**
<outras declarações>

FIGURA 26 - EXEMPLO DE INSERÇÃO PARA DISPARO DE TRIGGER



FONTE: A autora

3 COMPOSIÇÃO DO DATABASE TRIGGER

QUADRO 27 - CARACTERÍSTICAS DAS TRIGGERS

Parte	**Descrição**	**Valores possíveis**
Tempo do Trigger	Quando o trigger é ativado em relação ao disparo do evento.	- BEFORE - AFTER - INSTEAD OF (utilizado em views que não são modificáveis de outra forma)
Disparo do Evento	Qualquer operação de manipulação de dados sobre uma tabela causa disparo do trigger.	- Insert - Update - Delete
Tipo de Trigger	Quantas vezes o corpo da trigger pode ser executado.	- Comando - Linha
Corpo do Trigger	Que ação o trigger executa.	- Bloco PL/SQL completo

FONTE: A autora

4 DIFERENÇA ENTRE TRIGGERS E PROCEDURES

QUADRO 28 - DIFERENÇA DE TRIGGERS E PROCEDURES

Trigger	**Procedure**
Invocada implicitamente quando uma Manipulação de dados é realizada	Invocada explicitamente de uma aplicação ou procedure
COMMIT, ROLLBACK e SAVEPOINT São comandos proibidos dentro do corpo da trigger	COMMIT, ROLLBACK e SAVEPOINT são comandos permitidos dentro do corpo da procedure

FONTE: A autora

5 CRIANDO TRIGGERS DE COMANDO

Sintaxe:

```
CREATE OR REPLACE nome da trigger
tempo de disparo Evento de disparo OR outros ON nome tabela

        DECLARE
            variáveis, constantes, etc.
        BEGIN
            bloco pl/sql
        END;
```

Onde:
tempo de disparo: Indica quando o trigger deve ser ativado em relação ao evento: BEFORE / AFTER

evento do disparo: Identifica a operação de manipulação de dados que causará o disparo do trigger: INSERT / UPDATE / DELETE

nome da tabela: nome da tabela associada ao trigger

bloco pl/SQL: é o corpo do trigger que define a ação executada pelo trigger.

➔ EXEMPLO 1

Restringir a inserção de novos fornecedores a dias úteis e em horário comercial.

```
Create or replace trigger Rotina_Fornecedor
before insert on fornecedor
begin
  if (to_char(sysdate,'dy') in ('SAT','SUN')) or
    (to_char(sysdate,'hh24') not between 10 and 18) then
     raise_application_error (-20500,
         'Você só pode inserir novo Fornecedor em horário Comercial');
  end if;
end;

SQL> @TrigForn.Sql
 11  /

Trigger created

SQL> insert into fornecedor
    2  (cod_fornec, dsc_razao_social, num_cgc, num_cep, cod_cidade, dsc_
endereco,
  3  cod_tipo_fornec,tipo_fornec) values
  4  (12,'ffssdds',1111,1111,1,'sdsdds',2,2);

insert into fornecedor
            *
ERROR at line 1:
ORA-20500: Você só pode inserir novo Fornecedor em horário Comercial
ORA-06512: at line 4
ORA-04088: error during execution of trigger 'INFBLU.ROTINA_
FORNECEDOR'
```

6 EVENTOS DE DISPARO DA TRIGGER

É possível combinar diversos eventos de acionamento em um, utilizando os predicados condicionais INSERTING, UPDATING, DELETING no corpo da trigger.

➔ EXEMPLO 2

```
Create or replace trigger Rotina_Fornecedor
before delete or insert or update on fornecedor

Begin
if      (to_char(sysdate,'dy') in ('SAT','SUN')) or
        (to_char(sysdate,'hh24') not between 10 and 18) then

if deleting then
raise_application_error (-20500,
'Você só pode deletar novo Fornecedor em Horário
Comercial');

elsif inserting then
raise_application_error (-20500,
'Você só pode inserir novo Fornecedor em Horário
Comercial');

elsif updating then
raise_application_error (-20500,
'Você só pode atualizar novo Fornecedor em Horário
Comercial');

end if;
end if;
end;

SQL> @TrForn.Sql
 25  /
Trigger created.
```

7 CRIANDO TRIGGERS DE LINHA

Sintaxe:

```
CREATE OR REPLACE nome do trigger
Tempo_de_disparo Evento_de_disparo OR outros ON nome tabela
        FOR EACH ROW      WHEN restrição
        DECLARE
            variáveis,constantes,etc.
        BEGIN
            bloco pl/sql
        END;
```

Dentro de um trigger de linha, referencie o valor de uma coluna antes dos dados serem alterados prefixando-os com o qualificador OLD, e referencie seu valor depois dos dados serem alterados prefixando-os com o qualificador NEW.

QUADRO 29 - OPERAÇÕES COM TRIGGERS

Operação	**Valor Antigo (OLD)**	**Valor Novo (NEW)**
Insert	NULL	Valor inserido
Update	Valor antes da alteração	Valor alterado
Delete	Valor antes da exclusão	Null

FONTE: A autora

- Usar esses qualificadores dentro de triggers de linha BEFORE e AFTER.
- Não use estes qualificadores (old/new) dentro de triggers de comandos.
- Prefixe estes qualificadores com dois pontos (:) em todo comando SQL e PL/SQL, não na definição do trigger.
- Não é necessário definir os qualificadores OLD e NEW, e eles ficam locados para cada inserção, alteração ou exclusão.

➔ EXEMPLO

Identificar produtos de Fabricação Interna após a inserção de dados na tabela produtos.

```
Create or Replace trigger T_Pas_direta
after insert on produto
for each row
begin
if :new.cod_produto = 99 then
raise_application_error(-20220,'Produto de Fabricação Interna');
end if;
end;

1  insert into produto
2  (cod_produto,dsc_produto,dsc_unidade, vlr_unit)
3* Values (99,'Cabo para Martelo', 'Unidade', 1.99)
SQL> /
insert into produto
            *
ERROR at line 1:
ORA-20220: Produto de Fabricação Interna
ORA-06512: at "SCOTT.T_PAS_DIRETA", line 3
ORA-04088: error during execution of trigger 'SCOTT.T_PAS_DIRETA'
```

8 CLÁUSULA WHEN

Restringe a ação de um trigger para aquelas linhas que satisfaçam uma certa condição. Dentro da cláusula WHEN os prefixos OLD e NEW não podem ser precedidos de dois pontos. Não é permitido o uso da cláusula WHEN em triggers de comando.

➔ EXEMPLO

Na atualização e inserção da tabela item_pedido, calcula o valor unitário do produto de código 2 com acréscimo de 10%.

```
create or replace trigger atualiza_valor
before  insert or update of vlr_unit on item_pedido
for each row
when (new.cod_produto = 2)
begin
:new.vlr_unit := :new.vlr_unit + (:new.vlr_unit * 0.10);
end;

SQL> @AtVlr.Sql
  8  /

Trigger created.
```

```
SQL>  insert into item_pedido
  2       (num_pedido,cod_produto,num_solic,qtd_pedido,vlr_unit)
  3       values
  4       (1,2,1,10,10);

1 row created.

SQL> select * from item_pedido;

NUM_PEDIDO COD_PRODUTO NUM_SOLIC QTD_PEDIDO  VLR_UNIT
---------- ---------- --------- ---------- ---------
         2          2         1          2     50.78
         3          2         1          3    250.56
         3          3         1          2    536.32
         4          4         1          4   1050.23
         4          3         1          3     78.23
         4          2         1          4     12.33
         5          2         2          2    545.12
         5          4         2          2    256.66
         5          6         2          6      89.2
         6          1         3          2     88.26
         6          5         3          2    257.12
         7          1         4        150     78.15
         7          2         4          2   4056.56
         7          3         4          2    455.26
         7          5         4          3    789.65
         7          6         4         15    798.54
         7          4         4          2     10.12
         8          2         6        200   4998.23
         1          2         1         10        11

19 rows selected.
```

No momento da inserção, o Trigger acrescentou 10% ao valor unitário do item que foi inserido.

9 TRIGGERS DE COMANDO x LINHA

Implementar rotinas com triggers de comando para:

- Prevenir operações com dados inválidos.
- Auditoria de operações de dados.

- Controlar a segurança de operações de dados.
- Inicializar e resetar variáveis globais e flags.

Implementar rotinas com triggers de linhas para:

- Prevenir dados inválidos dentro de linhas.
- Auditoria de linhas e valores afetados por operações de dados.
- Controle dos dados e integridade referencial.
- Usar variáveis globais e flags para controlar operações de dados.
- Calcular valores derivados transparentemente.
- Alterar dados e executar funções implicitamente.
- Replicar tabelas.

10 CRIAÇÃO E DELEÇÃO DE TRIGGERS

QUADRO 30 - COMANDOS PARA MANIPULAR TRIGGERS

Tarefa	**Comando**
Criar um novo trigger	CREATE TRIGGER
Alterar um trigger existente	CREATE OR REPLACE TRIGGER
Excluir um trigger	DROP TRIGGER

FONTE: A autora

Observações

Há algumas diferenças importantes entre um database trigger e procedures.

11 DATABASE TRIGGER

- Escrever o comando CREATE TRIGGER em um arquivo texto.

- Rodar o comando CREATE TRIGGER do arquivo texto.

a) Se a compilação do trigger for um sucesso, o código fonte somente estará armazenado dentro do database, mas o p-code não.

b) Se a compilação do trigger falhou, nem o código fonte e nem os erros de sintaxe são escritos no dicionário de dados, mas os erros são comunicados interativamente.

Para que um trigger seja ativado, ele deve ter sido compilado antes.

PROCEDURE ARMAZENADA

Escrever o comando CREATE PROCEDURE em um arquivo texto.

Rodar o comando CREATE PROCEDURE do arquivo texto.

a) Se a compilação da procedure for um sucesso, o código fonte e o p-code estarão armazenados dentro do database.

b) Se a compilação da procedure falhou, o código fonte e os erros de sintaxe são escritos no dicionário de dados.

Quando a procedure é alterada, ela é normalmente recompilada (mecanismo automático).

Recompilar a procedure manualmente, se necessário.

12 HABILITANDO E DESABILITANDO TRIGGERS

Habilitando um database trigger:

ALTER TABLE nome trigger ENABLE

Habilitando todos os database trigger em relação a uma tabela:

ALTER TABLE nome table ENABLE ALL TRIGGERS

Desabilitando todos os database triggers em relação a uma tabela:

ALTER TABLE nome table DISABLE ALL TRIGGERS

13 GERENCIANDO TRIGGERS

Gerenciar o desenvolvimento de um trigger com comandos e estratégias que são um pouco diferentes daqueles usados para procedures.

QUADRO 31 - GERENCIAMENTO DE TRIGGERS

Tarefa	**Trigger**	**Procedure**
Documentar	Examinar a view USER_TRIGGERS do dicionário de dados.	Examinar as views USER_OBJECTS e USER_SOURCE do dicionário de dados.
Debug	Usar a procedure DBMS_OUTPUT e testar a operação de disparo.	Usar a procedure DBMS_OUTPUT.
Controlar a segurança para o desenvolvedor	Obter privilégios para criar a trigger, para acessar objetos referenciados pelo trigger e para alterar a tabela associada.	Obter privilégios para criar a procedure e para acessar objetos referenciados pela Procedure.
Controlar a segurança para o usuário	Nenhum privilégio especial	Obter privilégio EXECUTE para a procedure

FONTE: A autora

Privilégios necessários para desenvolver Triggers

QUADRO 32 - PRIVILÉGIOS PARA O DESENVOLVIMENTO

Operação	**Privilégio Requerido**	**Outros Privilégios**
CREATE	CREATE TRIGGER ou CREATE ANY TRIGGER.	Acessar todos os objetos referenciados pelo trigger e ter privilégio de acesso à tabela associada ou privilégio ALTER sobre a tabela associada ou privilégio ALTER ANY TABLE.
CREATE OR REPLACE	Todos os privilégios owner do trigger ou CREATE TRIGGER ou CREATE ANY TRIGGER.	Idem.
DROP	Todos os privilégios owner do trigger ou DROP ANY TRIGGER.	Não tem outros privilégios.

FONTE: A autora

14 DUAS REGRAS PARA LER E GRAVAR DADOS USANDO TRIGGER

Regra 1

Não podem ser alterados dados de chaves primárias, chaves estrangeiras, ou de chaves únicas de uma tabela. Em outra manipulação de dados, com uma coluna não chave da tabela, o trigger funciona sem restrições.

➔ EXEMPLO

```
create or replace trigger atualiza_Cascata
after update of cod_depto on depto
for each row
begin
update solic_compra
set solic_compra.cod_depto = :new.cod_depto
where solic_compra.cod_depto       = :old.cod_depto;
end;

SQL> @AtCasc.Sql
  9  /

Trigger created.

SQL>            update depto
  2      set cod_depto = 8
  3      where cod_depto = 1;

update depto
       *
ERROR at line 1:
ORA-04091: table SCOTT.DEPTO is mutating, trigger/function may not see it
ORA-06512: at "SCOTT.ATUALIZA_CASCATA", line 2
ORA-04088: error during execution of trigger 'SCOTT.ATUALIZA_CASCATA'
```

Regra 2

Não ler dados de uma tabela mutante, porque ela é afetada, direta ou indiretamente, pelo evento disparado, uma tabela que está sendo alterada.

➔ EXEMPLO

Verificar na tabela de itens qual o maior valor unitário. Enviar um erro ao usuário na inserção dos valores maiores do que aquele encontrado.

```
create or replace trigger Valor_maior
after insert or update on item_pedido
for each row
declare
v_cod_produto number;
v_vlr_unit number;
begin
select max(vlr_unit)
into v_vlr_unit
from item_pedido;
if :new.vlr_unit  > v_vlr_unit then
raise_application_error(-20100,'Valor Unitario com problemas');
end if;
end;

SQL> @MaxVlr.Sql
 15  /

Trigger created.

SQL> insert into item_pedido
  2      (num_pedido,cod_produto,num_solic,qtd_pedido,vlr_unit)
  3      values
  4      (1,2,1,10,600);

insert into item_pedido
            *
ERROR at line 1:
ORA-04091: table SCOTT.ITEM_PEDIDO is mutating, trigger/function may
not see it
ORA-06512: at line 5
ORA-04088: error during execution of trigger 'INFBLU.VALOR_MAIOR'
```

A falha ocorreu porque a alteração disparou o trigger da tabela que está sendo alterada, neste caso não é possível realizar a leitura.

15 DERIVAÇÃO DE DADOS USANDO UM TRIGGER BEFORE

➔ EXEMPLO

Criação de uma trigger que adiciona 10% aos valores unitários dos produtos de código 2 a cada atualização ou inserção de dados.

```
Create or replace trigger Novo_valor
BEFORE insert or update on item_pedido
for each row
when (new.cod_produto = 2)
begin
  :new.vlr_unit := :old.vlr_unit + (:new.vlr_unit * 0.10);
end;

SQL> @NewVlr.Sql
  7  /
Trigger created.

SQL> select * from item_pedido
  2       where cod_produto = 2;

NUM_PEDIDO COD_PRODUTO NUM_SOLIC QTD_PEDIDO  VLR_UNIT
---------- ---------- --------- ---------- ---------
         2          2         1          2     50.78
         3          2         1          3    250.56
         4          2         1          4     12.33
         5          2         2          2    545.12
         7          2         4          2   4056.56
         8          2         6        200   4998.23
         1          2         1         10        11

7 rows selected.

SQL> update item_pedido set qtd_pedido  = qtd_pedido + 1         where  cod_
produto         = 2;

7 rows updated. <neste momento o Trigger Novo_Valor crescentou 10% para
o valor unitário de todos os pedidos que contenham o produto de código 2>.

SQL>  select * from item_pedido
  2       where cod_produto = 2;
```

```
NUM_PEDIDO COD_PRODUTO NUM_SOLIC QTD_PEDIDO  VLR_UNIT
---------- ---------- --------- ---------- ---------
         2          2         1          3     55.86
         3          2         1          4    275.62
         4          2         1          5     13.56
         5          2         2          3    599.63
         7          2         4          3   4462.22
         8          2         6        201   5498.05
         1          2         1         11      12.1

7 rows selected.
```

Observações

- O qualificador NEW pode aparecer do lado esquerdo de um comando de assinalamento PL/SQL dentro do corpo de um trigger de linha BEFORE (:=).

- O qualificador NEW não pode aparecer do lado esquerdo de uma coluna dentro de um trigger de linha AFTER, ou dentro de um trigger de comando AFTER.

- O qualificador OLD nunca deve aparecer do lado esquerdo nem direito de coluna em triggers de comando (só para triggers de linhas).

16 ALTERAR DADOS EM UMA TABELA ASSOCIADA A UM TRIGGER SEM RESTRIÇÕES

➔ EXEMPLO

Insere na tabela "Salva_Item_Pedido" (que possui os mesmos campos da tabela Item_Pedido), o registro que será excluído da tabela Item_Pedido.

```
SQL> create table SALVA_ITEM_PEDIDO as select * from ITEM_PEDIDO
where 1=2;

Table created.

create or replace trigger Salva_pedido_ant
after delete on item_pedido
for each row
begin
  insert into salva_item_pedido
```

```
(num_pedido
,cod_Produto
,num_solic
,qtd_pedido
,vlr_unit)
values (:old.num_pedido,:old.cod_produto,
:old.num_solic, :old.qtd_pedido,
:old.vlr_unit);
end;

SQL> @Salva.Sql
 17  /

Trigger created.

SQL> delete from item_pedido
  2     where cod_produto    = 2
  3     and      num_solic       = 1
  4     and      num_pedido   = 2;

1 row deleted.
```

Momento em que o trigger inclui o registro excluído da tabela item_pedido na tabela Salva_Item_Pedido.

```
SQL> select * from salva_item_pedido;

NUM_PEDIDO COD_PRODUTO NUM_SOLIC QTD_PEDIDO   VLR_
UNIT
---------- ---------- --------- ---------- ---------
         2          2         1          3     55.86
```

Observação:

- Não há restrições na alteração de dados de uma tabela não relacionada, neste caso, o insert na tabela salva_item_pedido.

17 REGRAS PARA LEITURA E GRAVAÇÃO USANDO TRIGGERS – RESUMO

QUADRO 33 - RESUMO DE LEITURA E GRAVAÇÃO DE TRIGGERS

Operação	Caso	Validade
Gravação	Dados de tabela que não está sendo alterada, inserida ou excluída.	Permitido
	Colunas não chave de uma tabela usando INSERT, UPDATE ou DELETE.	Permitido
	Alteração de colunas chave (primary key, foreign key e unique key) de tabelas que contenham constraints.	Não permitido
	Colunas não alteradas de uma tabela com trigger usando comandos PL/SQL.	Permitido somente numa trigger de linha BEFORE
	Colunas alteradas de uma tabela com trigger usando comandos PL/SQL (derivação)	Permitido somente numa trigger de linha BEFORE
Leitura	De uma tabela não relacionada usando SELECT.	Permitido
	De uma tabela relacionada e não mutante usando select.	Permitido
	De uma tabela mutante, usando select.	Não permitido
	De tabela não mutante que possui trigger usando comandos PL/SQL.	Permitido

FONTE: A autora

18 APLICAÇÃO DE TRIGGERS

QUADRO 34 - APLICAÇÃO DAS TRIGGERS

Situação	Implementado pelo servidor	Implementado por triggers
Segurança	Permissão para acessar tabela individualmente ou por grupos de usuários.	Permissão para acessar tabelas baseado no valor dos dados (WHEN).
Auditoria	Trilha operações nas tabelas.	Trilha valores das operações de dados nas tabelas (NEW e OLD).
Integridade de dados	Declara integridade das constraints na criação de objetos.	Implementa regras de integridade.
Integridade referencial	Implementa funcionamento padrão usando constraints.	Implementa funcionamento não Standard.
Replicação de tabelas	Copia objetos usando SNAPSHOTS.	Copia tabelas usando comandos Sincronizadamente.
Dado Derivado	Calcula valores derivados Manualmente.	Calcula valores derivados Automaticamente.
Log de eventos	Executa operações explicitamente através da montagem de rotinas.	Executa operações transparentemente.

FONTE: A autora

19 APLICAR TRIGGERS PARA EXAMINAR VALORES – AUDITORIA

Examinar objetos dentro do server:

- Examinar dados recuperados, dados manipulados e comandos de definição de dados.
- Gravar o caminho para examinar manutenções numa tabela.
- Gerar registros uma vez por sessão ou uma vez por tentativa de acesso, tentativas bem-sucedidas, tentativas malsucedidas ou ambas.
- Habilitar e desabilitar Audit dinamicamente.

➔ EXEMPLO

Trilhar todas as operações bem-sucedidas sobre a tabela PRODUTO.

```
SQL> Audit Insert, Update, Delete
2       on Produto
3       by Access
4       Whenever Successful;

Audit succeeded.
```

20 APLICAR TRIGGERS PARA FORÇAR A INTEGRIDADE DOS DADOS

Restrição a Nível de Server

- Restrições são incorporadas às tabelas para garantir a integridade de dados (contraints com check).
- Forçar regras simples de integridade de dados.
- Não nulos, únicos, chave primária e chave estrangeira.
- Produzir valores default e constantes.
- Forçar uma condição estática, que deve ser válida.

21 FORÇAR A INTEGRIDADE DOS DADOS USANDO TRIGGERS

Desenvolvendo triggers para manipular regras mais complexas, que não podem ser incluídas no momento de criação da tabela.

- Forçar checks de integridade de dados.
- Produzir valores default para variáveis.
- Forçar condições dinamicamente ou atribuições.

22 APLICAR TRIGGERS PARA FORÇAR A INTEGRIDADE REFERENCIAL

➔ EXEMPLO

Quando um departamento é removido da tabela DEPT, realizar a exclusão em cascata das linhas correspondentes na tabela SOLIC_COMPRA:

```
SQL> alter table solic_compra add
        2       constraint solic_depto_fk
        3       foreign key (cod_depto) references depto (cod_depto)
        4       on delete cascade;
```

Forçar integridade referencial usando um trigger:

➔ EXEMPLO

```
SQL> create or replace trigger CASCADE_UPDATES
        2       after update of cod_depto on depto
        3       for each row
        4        begin
        5                update solic_compra
        6                set solic_compra.cod_depto   =:new.cod_Depto
        7                where solic_compra.cod_Depto        = :old.cod_depto;
        8        end;
```

23 APLICAR TRIGGERS PARA DERIVAR DADOS

Cálculo de dados derivados através do server

- Calcular colunas derivadas assincronamente, com intervalos definidos pelo usuário.
- **Armazenar valores derivados somente dentro de tabelas na base de dados, por exemplo, SELECT.**
- Modificar dados em uma passagem para a base de dados e calcular dados derivados em um segundo passo.

Derivar dados com triggers

- Calcular colunas derivadas sincronamente em tempo real.
- Armazenar valores derivados dentro de tabelas na base de dados ou dentro de variáveis globais de packages.
- Modificar dados e calcular dados derivados em uma única intervenção na base de dados.

24 APLICAR TRIGGERS PARA REPLICAR TABELAS

Conceito de Snapshots

São cópias de apenas leitura, ou de uma tabela completa, ou de parte das suas linhas (snapshot simples), ou de um conjunto de tabelas, visões ou linhas, utilizando visões, agrupamentos e critérios de seleção (snapshots complexos).

OBS.: Um problema com um registro de snapshot é que ele mantém uma cópia de cada uma das alterações na linha. Assim se uma linha sofrer 200 alterações entre uma atualização e a próxima, haverá 200 entradas no registro de Snapshots que será aplicado ao snapshot na atualização.

Copiar tabelas com snapshots do server

- Manter cópias de tabelas automaticamente com snapshots, particularmente em nós remotos, usando os recursos do server.
- Copiar tabelas assincronamente, em intervalos definidos pelo usuário.
- O snapshot pode apenas ler as tabelas.
- Melhorar a performance da manipulação de dados na tabela master, particularmente se a rede falhar.

Replicar tabelas com triggers

- Copiar tabelas no momento da manutenção, em tempo real.
- Usualmente, basear as réplicas sobre uma única tabela máster.
- Ler de réplicas e fazer manutenção nelas.

25 BENEFÍCIOS DE TRIGGERS DE BASE DE DADOS

Além das vantagens gerais de procedures armazenadas, os triggers de base de dados possuem mais benefícios, principalmente complementando a capacidade de atuação do servidor.

Melhorar a segurança dos dados

- Verifica valores baseados na segurança dos checks para manipulação de dados.
- Verifica valores baseados em auditorias para manipulação de dados.
- Replicação de tabelas.
- Auditoria de dados pró-ventos em tabelas.

Melhorar a integridade de dados

- Força condições de integridade de dados dinamicamente a nível de base de dados.
- Força condições de integridade referencial.
- Garante que operações de dados relacionadas sejam executadas implicitamente.

RESUMO DO TÓPICO 3

Neste tópico, você aprendeu:

- Que utilizar triggers é uma das maneiras mais práticas de implementar rotinas para garantir a integridade de dados ou de operações.
- Que triggers são rotinas ou procedures que são utilizadas quando um comando INSERT, UPDATE ou DELETE são executados em tabelas ou em visões. As triggers são disparadas quando estes comandos são executados.
- Que as triggers são disparadas automaticamente sem a interferência do usuário.
- Que a principal aplicação das triggers é a criação de consistências e restrições de acesso ao Banco de Dados, assim como as rotinas de segurança.

AUTOATIVIDADE

1. Exemplifique o uso do tempo before e after em uma trigger. Cite os eventos presentes em uma trigger.

2. Crie as tabelas abaixo:

```
create table N3_FUNC(COD_FUNC      NUMBER(5) not null,
                     NOM_FUNC      VARCHAR2(100) not null,
                     QTD_ANOS_SERV NUMBER(10),
                     VAL_SAL       NUMBER(10,2));

create table N3_FUNC_INTERM (COD_FUNC NUMBER(05) NOT NULL
PRIMARY KEY,
                     NOM_FUNC VARCHAR2(100) NOT NULL,
                     QTD_ANOS_SERV  NUMBER(10),
                     VAL_SAL   NUMBER(10,2),
DAT_ATUALZ  DATE,
TIPO_OPER   VARCHAR2(01),
NOM_USUAR   VARCHAR2(100));
```

Passos para a execução do exercício:

- Insira 5 linhas de registro na tabela N3_FUNC. Para o campo qtd_anos_servic, considerar apenas os valores de 1 a 5 anos.
- Crie uma trigger nesta tabela para toda vez que alterar, incluir ou excluir registros inserir na tabela N3_FUNC_INTERM, gravando a data atual e seu nome no campo usuário. Para tipo de operação = 'I' para inserção, 'A' para alteração e 'D' para exclusão.

REFERÊNCIAS

BATISTA, Ligia Flávia Antunes. Normalização. 2015. Disponível em: <inf.cp.cefetpr.br/ligia/material/bd1/normalizacao.ppt>. Acesso em: 07 jan. 2015.

BERGAMASCHI, S. Modelos de gestão da terceirização de tecnologia da informação: um estudo exploratório. 2004. 197 f. Tese (Doutorado em Administração) – Faculdade de Economia, Administração e Contabilidade, Universidade de São Paulo, USP, São Paulo, 2004.

CODD, T. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, v. 13, n. 6, 1970.

COUGO, Paulo Sérgio. Modelagem conceitual e projeto de banco de dados. Rio de Janeiro: Campus, 1997.

DATE, C. J. Introdução a sistemas de bancos de dados. Rio de Janeiro: Campus, 2002.

HEUSER, Carlos Alberto. Projeto de banco de dados. Porto Alegre: Sagra Luzzatto, 2004.

LAUDON, K.; LAUDON, J. P. Sistemas de informação gerenciais. 7. ed. São Paulo: Pearson, 2009.

_____. Gerenciamento de sistemas de informação. 3. ed. LTC: Rio de Janeiro, 2001.

MELLO, Ronaldo Santos. Página do professor. Disponível em: <http://www.inf.ufsc.br/~ronaldo/>. Acesso em: 23 jan. 2015.

O´BRIEN, J. A. Sistemas de informação. 2. ed. São Paulo: Saraiva, 2007.

PRESSMAN, R. S. Engenharia de Software. 6. ed. São Paulo: McGrawHill, 2006.

REYNOLDS, G. W. Princípios de sistema de informação. 4. ed. Rio de Janeiro: LTC, 2002.

SANTOS, Marilde. Normalização de Dados. 2014. Disponível em: <www.dc.ufscar.br/~marilde/Graduacao/PrimeiroSemestre/2006/Transparencias/Normalizacao.ppt>. Acesso em: 07 dez. 2014.

SHEDROFF, N. Information Design. Cambridge, Mass: MIT Press, 1999.

SILBERSCHATZ, A; KORTH, H. Sistema de banco de dados. São Paulo: Makron Books, 2004.

STAIR, R. M. Princípios de Sistema de Informação. 2. ed. Rio de Janeiro: LTC, 1998.

STEINBUHLER, K. E-Business e E-Commerce para Administradores. São Paulo: Pearson, 2004.

ANOTAÇÕES