

ORACLE®

SQL

Alexandre Costa Vieira
alexandre.costa@racionalenergia.com.br
(62) 9607-1555

Conteúdo

| | |
|-------------------------------------------|-----------|
| PREFÁCIO | 6 |
| NOÇÕES DE BANCO DE DADOS..... | 7 |
| O QUE É UM BANCO DE DADOS | 7 |
| UTILIZAÇÃO | 7 |
| APRESENTAÇÃO DOS DADOS | 8 |
| VISÃO DE NEGÓCIO..... | 8 |
| MODELOS DE BASE DE DADOS | 8 |
| APLICAÇÕES DE BANCOS DE DADOS | 9 |
| APLICATIVO DE BANCO DE DADOS..... | 9 |
| TRANSAÇÃO | 10 |
| <i>Atomicidade</i> | 10 |
| <i>Consistência</i> | 10 |
| <i>Isolamento</i> | 10 |
| <i>Durabilidade</i> | 10 |
| SEGURANÇA EM BANCO DE DADOS..... | 11 |
| FUNÇÕES INTERNAS COMUNS EM BDS | 11 |
| NOÇÕES DE BANCOS RELACIONAIS | 11 |
| O MODELO RELACIONAL..... | 11 |
| BANCO DE DADOS EXEMPLO..... | 12 |
| EXEMPLO: CLIENTE | 13 |
| 13 REGRAS DE TED CODD..... | 14 |
| NOÇÕES DE CONJUNTO | 16 |
| ORIGEM..... | 16 |
| CONJUNTO..... | 17 |
| RELAÇÕES | 17 |
| ALGUNS CONCEITOS | 17 |
| ALGUMAS NOTAÇÕES PARA CONJUNTOS | 18 |
| SUBCONJUNTOS..... | 19 |
| ALGUNS CONJUNTOS ESPECIAIS..... | 19 |
| REUNIÃO DE CONJUNTOS..... | 19 |
| INTERSEÇÃO DE CONJUNTOS..... | 19 |
| PROPRIEDADES DOS CONJUNTOS | 20 |
| DIFERENÇA DE CONJUNTOS | 21 |
| COMPLEMENTO DE UM CONJUNTO..... | 22 |
| LEIS DE AUGUSTUS DE MORGAN | 22 |
| DIFERENÇA SIMÉTRICA..... | 23 |
| A LINGUAGEM SQL..... | 24 |
| INTRODUÇÃO AO SQL | 24 |
| CARACTERÍSTICAS DO SQL..... | 24 |
| O CONJUNTO DE COMANDOS SQL | 25 |
| ESCREVENDO COMANDOS SQL..... | 26 |
| O COMANDO SELECT | 26 |
| OUTROS ITENS DA CLAUSULA SELECT | 27 |
| EXPRESSÕES ARITMÉTICAS | 28 |
| COLUNAS SINÔNIMAS..... | 29 |
| O OPERADOR DE CONCATENAÇÃO..... | 30 |
| LITERAIS | 30 |
| MANUSEANDO VALORES NULOS..... | 31 |

| | |
|--------------------------------------------------------------------|-----------|
| FUNÇÃO NVL | 31 |
| PREVENINDO A SELEÇÃO DE LINHAS DUPLICADAS | 32 |
| A CLAUSULA DISTINCT | 32 |
| A CLAUSULA ORDER BY | 33 |
| <i>Padrão da Ordenação dos Dados</i> | 33 |
| <i>Invertendo o padrão de ordenação</i> | 33 |
| <i>Ordenação por várias colunas</i> | 34 |
| A CLAUSULA WHERE | 35 |
| <i>Operadores Lógicos</i> | 35 |
| <i>Alfanuméricos e Datas na clausula WHERE</i> | 35 |
| <i>Comparando uma coluna com outra coluna na mesma linha</i> | 36 |
| <i>Operadores SQL</i> | 36 |
| <i>O Operador BETWEEN</i> | 37 |
| <i>O Operador IN</i> | 37 |
| <i>O Operador LIKE</i> | 37 |
| <i>Operador IS NULL</i> | 38 |
| <i>Expressões Negativas</i> | 38 |
| <i>Pesquisando Dados com Múltiplas Condições</i> | 40 |
| <i>Precedência dos Operadores</i> | 42 |
| RESUMO DO SELECT. | 42 |
| EXERCÍCIO 1 - INTRODUÇÃO AO SQL, O COMANDO SELECT | 43 |
| FUNÇÕES..... | 46 |
| INTRODUÇÃO A FUNÇÕES | 46 |
| FUNÇÕES DE LINHA ÚNICA | 47 |
| FUNÇÕES ALFANUMÉRICAS E NUMÉRICAS..... | 47 |
| <i>Funções Alfanuméricas</i> | 47 |
| <i>REPLACE</i> | 52 |
| <i>Aninhamento de Funções</i> | 52 |
| FUNÇÕES NUMÉRICAS | 53 |
| <i>ROUND</i> | 53 |
| <i>TRUNC</i> | 53 |
| <i>CEIL</i> | 54 |
| <i>FLOOR</i> | 54 |
| <i>POWER</i> | 54 |
| <i>SQRT</i> | 55 |
| <i>SIGN</i> | 55 |
| <i>ABS</i> | 55 |
| <i>MOD</i> | 56 |
| EXERCÍCIO 2 - USANDO FUNÇÕES | 56 |
| FUNÇÕES DE DATA..... | 58 |
| <i>Armazenamento de Datas no ORACLE</i> | 58 |
| <i>Sysdate</i> | 58 |
| <i>Usando Operadores Aritméticos</i> | 59 |
| <i>MONTHS_BETWEEN</i> | 59 |
| <i>ADD_MONTHS</i> | 60 |
| <i>NEXT_DAY</i> | 60 |
| <i>LAST_DAY</i> | 60 |
| <i>TRUNC</i> | 61 |
| FUNÇÕES DE CONVERSÃO | 61 |
| <i>TO_CHAR</i> | 61 |
| FORMATOS DE DATA | 63 |
| FORMATOS NUMÉRICOS | 64 |
| <i>TO_NUMER</i> | 65 |
| <i>TO_DATE</i> | 65 |
| FUNÇÕES QUE ACEITAM VÁRIOS TIPOS DE ENTRADA DE DADOS..... | 66 |

| | |
|----------------------------------------------------------------|-----------|
| <i>DECODE</i> | 66 |
| <i>NVL</i> | 68 |
| <i>GREATEST</i> | 68 |
| <i>LEAST</i> | 68 |
| <i>VSIZE</i> | 69 |
| REVISANDO ANINHAMENTO DE FUNÇÕES | 69 |
| EXERCÍCIO 3 – MAIS FUNÇÕES | 70 |
| FUNÇÕES DE GRUPO | 72 |
| <i>GROUP BY</i> | 72 |
| USANDO FUNÇÕES DE GRUPO: | 72 |
| <i>AVG</i> | 72 |
| <i>MIN</i> | 73 |
| <i>COUNT</i> | 73 |
| A CLAUSULA GROUP BY | 73 |
| <i>Excluindo linhas quando estiver Usando o GROUP BY</i> | 74 |
| <i>Grupos dentro de Grupos</i> | 74 |
| FUNÇÕES DE GRUPO E RESULTADOS INDIVIDUAIS | 74 |
| A CLAUSULA HAVING | 76 |
| A ORDEM DAS CLAUSULAS NA DECLARAÇÃO SELECT. | 77 |
| EXERCÍCIO 4 - FUNÇÕES DE GRUPO | 77 |
| EXTRAINDO DADOS DE MAIS DE UMA TABELA..... | 79 |
| LIGAÇÕES (JOINS) | 79 |
| <i>Equi-Join</i> | 79 |
| USANDO TABELAS COM SINÔNIMOS..... | 80 |
| <i>Nom-Equi-Join</i> | 81 |
| REGRAS PARA LIGAÇÕES DE TABELAS | 82 |
| EXERCÍCIO 5 - SIMPLES LIGAÇÕES (JOIN)..... | 82 |
| OUTROS MÉTODOS DE LIGAÇÃO | 84 |
| LIGAÇÕES EXTERNAS (OUTHER JOIN) | 84 |
| LIGANDO UMA TABELA COM ELA MESMA | 85 |
| OPERADORES DE CONJUNTO | 86 |
| <i>UNION</i> | 86 |
| <i>INTERSECT</i> | 87 |
| <i>MINUS</i> | 87 |
| ORDER BY | 87 |
| <i>Regras Quando Usar Operadores de Conjuntos</i> | 88 |
| EXERCÍCIO 6 - OUTROS MÉTODOS DE LIGAÇÕES..... | 89 |
| SUB-PESQUISAS (SUBQUERIES)..... | 90 |
| ANINHAMENTO DE SUB-PESQUISAS | 90 |
| SUB-PESQUISAS DE LINHA ÚNICA | 91 |
| COMO SÃO PROCESSADAS AS SUB-PESQUISAS ANINHADAS? | 91 |
| SUB-PESQUISAS QUE RETORNA MAIS DE UMA LINHA..... | 92 |
| COMPARANDO MAIS DE UM VALOR | 93 |
| OPERADORES ANY OU ALL | 94 |
| CLAUSULA HAVING COM SUB-PESQUISAS ANINHADAS..... | 95 |
| ORDENANDO DADOS COM SUB-PESQUISAS | 95 |
| SUB-PESQUISAS ANINHADAS | 96 |
| LIMITES DE ALINHAMENTO | 96 |
| <i>Diretriz</i> | 96 |
| SUB-PESQUISAS CORRELATAS | 97 |
| A PESQUISA INTERNA..... | 98 |
| <i>Operadores</i> | 98 |

| | |
|----------------------------------------------------------------------|------------|
| OPERADOR EXISTS | 98 |
| POR QUE USAR UMA SUB-PESQUISA CORRELATA? | 99 |
| EXERCÍCIO 7 - SUB-PESQUISAS | 100 |
| LINGUAGEM DE MANIPULAÇÃO DOS DADOS | 102 |
| INSERT | 102 |
| <i>Inserindo Datas e informações de Horas.</i> | 103 |
| <i>Copiando linhas de Outra Tabela</i> | 103 |
| UPDATE | 103 |
| DELETE..... | 105 |
| COMMIT E ROLLBACK | 105 |
| PROCESSANDO TRANSAÇÕES..... | 106 |
| O QUE É UMA TRANSAÇÃO? | 106 |
| DESENHO E TERMOLOGIA DO BÁSICO BANCO DE DADOS RELACIONAL..... | 106 |
| DESENHO..... | 106 |
| SIMPLES DIAGRAMA DE ENTIDADE E RELACIONAMENTO..... | 107 |
| <i>Como ler o Diagrama.</i> | 107 |
| O DIAGRAMA DE TABELAS..... | 107 |
| <i>Diagrama de Tabelas:</i> | 109 |
| CHAVE ESTRANGEIRA(FOREIGN KEY) | 109 |
| COLUNAS OBRIGATÓRIAS E OPCIONAIS | 110 |
| LINGUAGEM DE DEFINIÇÃO DE DADOS E DICIONÁRIOS DE DADOS..... | 111 |
| ESTRUTURA DE DADOS ORACLE..... | 111 |
| CRIANDO UMA TABELA | 111 |
| <i>Diretriz para Nomear Tabelas.</i> | 111 |
| <i>Tipos de Colunas</i> | 112 |
| <i>Criando uma Tabela (CREATE TABLE)</i> | 113 |
| CLAUSULA CONSTRAINT..... | 114 |
| <i>Parâmetro das Restrições.</i> | 116 |
| CRIANDO UMA TABELA COM LINHAS DE OUTRA TABELA. | 116 |
| ALTERANDO UMA TABELA | 118 |
| EXCLUINDO UMA TABELA | 120 |
| O COMANDO COMMENT | 120 |
| O COMANDO RENAME..... | 120 |
| O DICIONÁRIO DE DADOS ORACLE | 121 |
| ACESSANDO O DICIONÁRIO DE DADOS..... | 121 |
| DICIONÁRIO DE DADOS TABELAS E VISÕES..... | 121 |
| TABELAS | 121 |
| SINÔNIMOS | 122 |
| A VISÃO DICTIONARY | 122 |
| EXERCÍCIO 8 - LINGUAGEM DE DEFINIÇÃO DE DADOS | 126 |
| RESPOSTAS DOS EXERCÍCIOS PROPOSTOS | 128 |
| EXERCÍCIO 1 - INTRODUÇÃO AO SQL, O COMANDO SELECT | 128 |
| EXERCÍCIO 2 - USANDO FUNÇÕES | 129 |
| EXERCÍCIO 3 – MAIS FUNÇÕES | 129 |
| EXERCÍCIO 4 - FUNÇÕES DE GRUPO..... | 130 |
| EXERCÍCIO 5 - SIMPLES LIGAÇÕES (JOIN)..... | 131 |
| EXERCÍCIO 6 - OUTROS MÉTODOS DE LIGAÇÕES..... | 132 |
| EXERCÍCIO 7 - SUB-PESQUISAS | 133 |
| EXERCÍCIO 8 - LINGUAGEM DE DEFINIÇÃO DE DADOS | 134 |

Prefácio

Este curso explora toda a potência da linguagem SQL. Uma básica introdução para concepção do Desenho do Banco de Dados Relacional o qual estamos discutindo.

PÚBLICO

Todos aqueles que estão tecnicamente envolvidos na criação do ORACLE aplicações ou manutenções desse aplicativo, Programadores, Administradores de Banco de Dados, Analistas e Programadores Sênior. Usuários finais quem deseja estender seus conhecimentos, pode achar o curso benéfico.

PRÉ-REQUISITOS

Um grau de conhecimento em computação é exigido. Uso de um Micro Computador é desejável.

OBJETIVOS

No final do curso os alunos estarão aptos para:

- Configurar e manter a estrutura de dados ORACLE.
- Efetuar complexas pesquisas.
- Completar básico entendimento de Relacional o princípio e terminologia.

MANEIRA DE APRESENTAÇÃO

Este curso é direcionado com combinações tradicionais de seções com médias projeções e seções de demonstrações usando a tela de um projetor. Os alunos terão ampla oportunidade para praticar os tópicos trazidos nas seções anteriores.

HORÁRIO

Este curso exige no mínimo 15 horas divididas em 5 dias, e mais 30 horas de dedicação extra curso para que o aluno possa assimilar os conceitos apresentados em sala de aula.

Noções de banco de dados

O que é um banco de dados

Banco de dados (ou base de dados), é um conjunto de registros dispostos em estrutura regular que possibilita a reorganização dos mesmos e produção de informação. Um banco de dados normalmente agrupa registros utilizáveis para um mesmo fim.

Um banco de dados é usualmente mantido e acessado por meio de um software conhecido como Sistema Gerenciador de Banco de Dados (SGBD). Normalmente um SGBD adota um modelo de dados, de forma pura, reduzida ou estendida. Muitas vezes o termo banco de dados é usado, de forma errônea, como sinônimo de SGDB.

O modelo de dados mais adotado hoje em dia é o modelo relacional, onde as estruturas têm a forma de tabelas, compostas por tuplas (linhas) e colunas.

Um Sistema de Gestão de Bases de Dados, (SGBD) não é nada mais do que um conjunto de programas que permitem armazenar, modificar e extrair informação de um banco de dados. Há muito tipos diferentes de SGBD. Desde pequenos sistemas que funcionam em computadores pessoais a sistemas enormes que estão associados a mainframes. Um Sistema de Gestão de Base 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 as componentes de um SGBD:

1. Linguagem de definição de dados (especifica conteúdos, estrutura a base de dados e define os elementos de dados);
2. Linguagem de manipulação de dados (para poder alterar os dados na base);
3. Dicionário de dados (guarde definições de elementos de dados e respectivas características – descreve os dados, quem os acede, etc.

Utilização

Os bancos de dados são utilizados em muitas aplicações, abrangendo praticamente todo o campo dos programas de computador. Os bancos de dados são o método de armazenamento preferencial e baseiam-se em tecnologias padronizadas de bancos de dados. Um banco de dados é um conjunto de informações com uma estrutura regular. Um banco de dados é normalmente, mas não necessariamente, armazenado em algum formato de máquina legível para um computador. Há uma grande variedade de bancos de dados, desde simples tabelas armazenadas em um único arquivo até gigantescos bancos de dados com muitos milhões de registros, armazenados em salas cheias de discos rígidos. Bancos de dados caracteristicamente modernos são desenvolvidos desde os anos da década de 1960.

Apresentação dos dados

A apresentação dos dados geralmente é semelhante à de uma planilha eletrônica, porém os sistemas de gestão de banco de dados possuem características especiais para o armazenamento, classificação, gestão da integridade e recuperação dos dados. Com a evolução de padrões de conectividade entre as tabelas de um banco de dados e programas desenvolvidos em linguagens como Java, Delphi, Visual Basic, C++ etc, a apresentação dos dados, bem como a navegação, passou a ser definida pelo programador ou o designer de aplicações. Como hoje em dia a maioria das linguagens de programação fazem ligações a bancos de dados, a apresentação destes tem ficado cada vez mais a critério dos meios de programação, fazendo com que os bancos de dados deixem de restringir-se às pesquisas básicas, dando lugar ao compartilhamento, em tempo real, de informações, mecanismos de busca inteligentes e permissividade de acesso hierarquizada.

Visão de negócio

Todas as organizações têm quantidades, por vezes, astronômicas de dados e informação que têm de armazenar. Contudo, o papel tem problemas ao nível da persistência (tempo e tipo de visualização) e da recuperação (validação e verificação), ou seja, dura pouco. Neste sentido, torna-se mais fácil encontrar a informação numa base de dados que recorre a uma das tecnologias de informação de maior sucesso. Ou seja, as bases de dados estendem a função do papel ao guardar a informação em computadores. Qualquer empresa que pretenda garantir um controle efetivo sobre todo o seu negócio, tem obrigatoriamente de recorrer a sistemas de gestão de bases de dados. O Microsoft Excel continua a ser uma ferramenta de controle extremamente poderosa porque consegue operacionalizar os dados e assim criar informação útil ao planeamento diário das empresas. Contudo, existem outro tipo de ferramentas, mais completas e com funcionalidades acrescidas que elevam para outros níveis, a capacidade operacional de gerar informação de valor para a organização.

Modelos de base de dados

O modelo plano (ou tabular) consiste de matrizes simples, bidimensionais, compostas por elementos de dados: inteiros, números reais, etc. Este modelo plano é a base das planilhas eletrônicas.

O modelo em rede permite que várias tabelas sejam usadas simultaneamente através do uso de apontadores (ou referências). Algumas colunas contêm apontadores para outras tabelas ao invés de dados. Assim, as tabelas são ligadas por referências, o que pode ser visto como uma rede. Uma variação particular deste modelo em rede, o modelo hierárquico, limita as relações a uma estrutura semelhante a uma árvore (hierarquia - tronco, galhos), ao invés do modelo mais geral direcionado por grafos.

Bases de dados relacionais consistem, principalmente de três componentes: uma coleção de estruturas de dados, nomeadamente relações, ou informalmente tabelas; uma coleção dos operadores, a álgebra e o cálculo relacionais; e uma coleção de restrições da integridade, definindo o conjunto consistente de estados de base de dados e de alterações

de estados. As restrições de integridade podem ser de quatro tipos: domínio (também conhecidas como type), atributo, relvar (variável relacional) e restrições de base de dados.

Diferentemente dos modelos hierárquico e de rede, não existem quaisquer apontadores, de acordo com o Princípio de Informação: toda informação tem de ser representada como dados; qualquer tipo de atributo representa relações entre conjuntos de dados. As bases de dados relacionais permitem aos utilizadores (incluindo programadores) escreverem consultas (queries) que não foram antecipadas por quem projetou a base de dados. Como resultado, bases de dados relacionais podem ser utilizadas por várias aplicações em formas que os projetistas originais não previram, o que é especialmente importante em bases de dados que podem ser utilizadas durante décadas. Isto tem tornado as bases de dados relacionais muito populares no meio empresarial.

O modelo relacional é uma teoria matemática desenvolvida por Edgar Frank Codd, matemático e pesquisador da IBM, para descrever como as bases de dados devem funcionar. Embora esta teoria seja a base para o software de bases de dados relacionais, muito poucos sistemas de gestão de bases de dados seguem o modelo de forma restrita ou a pé da letra – mais adiante nos aprofundaremos neste modelo - e todos têm funcionalidades que violam a teoria, desta forma variando a complexidade e o poder. A discussão se esses bancos de dados merecem ser chamados de relacional ficou esgotada com o tempo, com a evolução dos bancos existentes. Os bancos de dados hoje implementam o modelo definido como objeto-relacional.

Aplicações de bancos de dados

Sistemas Gerenciadores de Bancos de dados são usados em muitas aplicações, enquanto atravessando virtualmente a gama inteira de software de computador. Os Sistemas Gerenciadores de Bancos de dados são o método preferido de armazenamento/recuperação de dados/informações para aplicações multi-usuárias grandes onde a coordenação entre muitos usuários é necessária. Até mesmo usuários individuais os acham conveniente, entretanto, muitos programas de correio eletrônico e organizadores pessoais estão baseados em tecnologia de banco de dados standard.

Aplicativo de Banco de Dados

Um Aplicativo de Banco de dados é um tipo de software exclusivo para gerenciar um banco de dados. Aplicativos de banco de dados abrangem uma vasta variedade de necessidades e objectivos, de pequenas ferramentas como uma agenda, até complexos sistemas empresariais para desempenhar tarefas como a contabilidade.

O termo "Aplicativo de Banco de dados" usualmente se refere a softwares que oferecem uma interface para o banco de dados. O software que gerencia os dados é geralmente chamado de sistema gerenciador de banco de dados (SGBD) ou (se for embarcado) de "database engine".

Exemplos de aplicativos de banco de dados são Microsoft Visual FoxPro, Microsoft Access, dBASE, MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Informix, DB2, Caché e Sybase.

Transação

É um conjunto de procedimentos que é executado num banco de dados, que para o usuário é visto como uma única ação.

A integridade de uma transação depende de 4 propriedades, conhecidas como ACID.

Atomicidade

Todas as ações que compõem a unidade de trabalho da transação devem ser concluídas com sucesso, para que seja efetivada. Qualquer ação que constitui falha na unidade de trabalho, a transação deve ser desfeita (*rollback*). Quando todas as ações são efetuadas com sucesso, a transação pode ser efetivada (*commit*).

Consistência

Nenhuma operação do banco de dados de uma transação pode ser parcial. O status de uma transação deve ser implementado na íntegra. Por exemplo, um pagamento de conta não pode ser efetivado se o processo que debita o valor da conta corrente do usuário não for efetivado antes, nem vice-versa.

Isolamento

Cada transação funciona completamente à parte de outras estações. Todas as operações são parte de uma transação única. O princípio é que nenhuma outra transação, operando no mesmo sistema, pode interferir no funcionamento da transação corrente (é um mecanismo de controle). Outras transações não podem visualizar os resultados parciais das operações de uma transação em andamento.

Durabilidade

Significa que os resultados de uma transação são permanentes e podem ser desfeitos somente por uma transação subsequente. Por 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.

Na prática, alguns SGBDs relaxam na implementação destas propriedades buscando desempenho.

Controle de concorrência é um método usado para garantir que as transações sejam executadas de uma forma segura e sigam as regras ACID. Os SGBD devem ser capazes de assegurar que nenhuma ação de transações completadas com sucesso (*committed transactions*) seja perdida ao desfazer transações abortadas (*rollback*).

Uma transação é uma unidade que preserva consistência. Requeremos, portanto, que qualquer escalonamento produzido ao se processar um conjunto de transações concorrentemente seja computacionalmente equivalente a um escalonamento produzindo executando essas transações serialmente em alguma ordem. Diz-se que um sistema que garante esta propriedade assegura a seriabilidade.

Segurança em banco de dados

Os bancos de dados são utilizados para armazenar diversos tipos de informações, desde dados sobre uma conta de e-mail até dados importantes da Receita Federal.

Para tal existem diversos tipos, os quais variam em complexidade e sobretudo em segurança.

- Criptografia
- Senhas
- Backup
- Clientes
- Fotos

Funções internas comuns em BDs

- Tabelas
- Regras
- Procedimentos armazenados (mais conhecidos como stored procedures)
- Gatilho
- Default
- Visão
- Índice

Noções de bancos relacionais

O modelo relacional é um modelo de dados, adequado a ser o modelo subjacente de um Sistema Gerenciador de Banco de Dados (SGBD), que se baseia no princípio em que todos os dados estão guardados em tabelas (ou, matematicamente falando, relações). Toda sua definição é teórica e baseada na lógica de predicados e na teoria dos conjuntos.

O conceito foi criado por Edgar Frank Codd em 1970, sendo descrito no artigo "*Relational Model of Data for Large Shared Data Banks*". Na verdade, o modelo relacional foi o primeiro modelo de dados descrito teoricamente, os bancos de dados já existentes passaram então a ser conhecidos como (modelo hierárquico, modelo em rede ou Codasyl e modelo de listas invertidas).

O modelo relacional

O modelo relacional para gerência de bancos de dados (SGBD) é um modelo de dados baseado em lógica e na teoria de conjuntos.

Em definição simplificada, o modelo baseia-se em dois conceitos: conceito de entidade e relação - Uma entidade é um elemento caracterizado pelos dados que são

recolhidos na sua identificação vulgarmente designado por tabela. Na construção da tabela identificam-se os dados da entidade a atribuição de valores a uma entidade constrói um registro da tabela. A relação determina o modo como cada registro de cada tabela se associa a registros de outras tabelas.

Historicamente ele é o sucessor do modelo hierárquico e do modelo em rede. Estas arquiteturas antigas são até hoje utilizadas em alguns data centers com alto volume de dados, onde a migração é inviabilizada pelo custo que ela demandaria; existem ainda os novos modelos baseados em orientação ao objeto, que na maior parte das vezes são encontrados como kits em linguagem formal.

O modelo relacional foi inventado pelo Dr. Codd e subsequentemente mantido e aprimorado por Chris Date e Hugh Darwen como um modelo geral de dados. No Terceiro Manifesto (1995) eles mostraram como o modelo relacional pode ser estendido com características de orientação a objeto sem comprometer os seus princípios fundamentais.

A linguagem padrão para os bancos de dados relacionais, SQL, é apenas vagamente remanescente do modelo matemático. Atualmente ela é adotada, apesar de suas restrições, porque ela é antiga e muito mais popular que qualquer outra linguagem de banco de dados.

A principal proposição do modelo relacional é que todos os dados são representados como relações matemáticas, isto é, um subconjunto do produto Cartesiano de n conjuntos. No modelo matemático (diferentemente do SQL), a análise dos dados é feita em uma lógica de predicados de dois valores (ou seja, sem o valor nulo); isto significa que existem dois possíveis valores para uma proposição: verdadeira ou falsa. Os dados são tratados pelo cálculo relacional ou álgebra relacional.

O modelo relacional permite ao projetista criar um modelo lógico consistente da informação a ser armazenada. Este modelo lógico pode ser refinado através de um processo de normalização. Um banco de dados construído puramente baseado no modelo relacional estará inteiramente normalizado. O plano de acesso, outras implementações e detalhes de operação são tratados pelo sistema DBMS, e não devem ser refletidos no modelo lógico. Isto se contrapõe à prática comum para DBMSs SQL nos quais o ajuste de desempenho frequentemente requer mudanças no modelo lógico.

Os blocos básicos do modelo relacional são o domínio, ou tipo de dado. Uma tupla é um conjunto de atributos que são ordenados em pares de domínio e valor. Uma relvar (variável relacional) é um conjunto de pares ordenados de domínio e nome que serve como um cabeçalho para uma relação. Uma relação é um conjunto desordenado de tuplas. Apesar destes conceitos matemáticos, eles correspondem basicamente aos conceitos tradicionais dos bancos de dados. Uma relação é similar ao conceito de tabela e uma tupla é similar ao conceito de linha.

O princípio básico do modelo relacional é o princípio da informação: toda informação é representada por valores em relações (*relvars*). Assim, as *relvars* não são relacionadas umas às outras no momento do projeto. Entretanto, os projetistas utilizam o mesmo domínio em vários *relvars*, e se um atributo é dependente de outro, esta dependência é garantida através da integridade referencial.

Banco de dados exemplo

Um exemplo, bem simples da descrição de algumas tabelas e seus atributos:

Cliente(**ID Cliente**, ID Taxa, Nome, Endereço, Cidade, Estado, CEP, Telefone)

Pedido de compra(**Número do pedido**, ID Cliente, Fatura, Data do pedido, Data prometida, Status)

Item do pedido(**Número do pedido**, Número do item, Código do produto, Quantidade)

Nota fiscal(**Número da nota**, ID Cliente, Número do pedido, Data, Status)

Item da nota fiscal(**Número da nota**, Número do item, Código do produto, Quantidade vendida)

Neste projeto nós temos cinco *relvars*: Cliente, Pedido de compra, Item do pedido, Nota fiscal e Item da nota fiscal. Os atributos em negrito são chaves candidatas. Os itens sublinhados (em negrito ou não, chaves compostas) são as chaves estrangeiras.

Normalmente uma chave candidata é escolhida arbitrariamente para ser chamada de chave primária e utilizada com preferência sobre as outras chaves candidatas, que são então chamadas de chaves alternativas.

Uma chave candidata é um identificador único que garante que nenhuma tupla será duplicada; isto faz com que o relacionamento em algo denominado um multiconjunto, porque viola a definição básica de um conjunto. Uma chave pode ser composta, isto é, pode ser formada por vários atributos (chave composta). Abaixo temos um exemplo tabular da nossa variável exemplo Cliente; um relacionamento pode ser abstraído como um valor que pode ser atribuído a uma *relvar*.

Exemplo: cliente

| ID Cliente | ID Taxa | Nome | Endereço | [Mais campos...] |
|------------|-------------|-----------------|---------------------|------------------|
| 1234567890 | 555-5512222 | João Carlos | Rua bigodone, 120 | ... |
| 2223344556 | 555-5523232 | Dorotéia Santos | Avenida barbeiro,12 | ... |
| 3334445563 | 555-5533322 | Lisbela da Cruz | Rua dos bigodes,123 | ... |
| 4232342432 | 555-5325523 | E. F. Codd | Rua do gilete,51 | ... |

Se nós tentarmos inserir um novo cliente com o ID 1234567890, isto irá violar o projeto da *relvar* pois ID Cliente é uma chave primária e nós já temos um cliente com o número 1234567890. O DBMS deve rejeitar uma transação como esta e deve acusar um erro de violação da integridade.

As chaves estrangeiras são condições de integridade que garantem que o valor de um atributo é obtido de uma chave candidata de outra *relvar*, por exemplo na *relvar* Pedido o atributo ID Cliente é uma chave estrangeira. Uma união é uma operação que retorna a informação de várias *relvars* de uma vez. Através da união de *relvars* do exemplo acima podemos consultar no banco de dados quais são os clientes, pedidos e

notas. Se nós queremos apenas as tuplas de um cliente específico, podemos especificar isto utilizando uma condição de restrição. Se queremos obter todos os pedidos do cliente 1234567890, podemos consultar o banco de dados de forma que este retorne toda linha na tabela de Pedidos com ID Cliente igual a 1234567890 e agrupar a tabela de pedidos com a tabela de itens de pedido baseado no Número do pedido. Existe uma imperfeição no projeto de banco de dados acima. A tabela de notas contém um atributo número do pedido. Assim, cada tupla na tabela de notas terá um pedido, o que implica precisamente um pedido para cada nota, mas na realidade uma nota pode ser criada a partir de muitos pedidos, ou mesmo para nenhum pedido em particular. Adicionalmente um pedido contém um número de nota, implicando que cada pedido tem uma nota correspondente. Mas novamente isto não é verdadeiro no mundo real. Um pedido é às vezes pago através de várias notas, e às vezes pago sem um nota. Em outras palavras podemos ter muitas notas por pedido e muitos pedidos por nota. Isto é um relacionamento vários-para-vários entre pedidos e notas. Para representar este relacionamento no banco de dados uma nova tabela deve ser criada com o propósito de especificar a correspondência entre pedidos e notas:

PedidoNota(Número do pedido, Número da nota)

Agora, um pedido tem um relacionamento um-para-vários para a tabela PedidoNota, assim como o Cliente tem para a tabela de pedidos. Se quisermos retornar todas as notas de uma pedido específico, podemos consultar no banco de dados todos os pedidos cujo Número do pedido é igual ao Número do pedido na tabela PedidoNota, e onde o Número da nota na tabela PedidoNota é igual à Número da nota na tabela Notas. A normalização de banco de dados é normalmente realizada quando projeta-se um banco de dados relacional, para melhorar a consistência lógica do projeto do banco de dados e o desempenho transacional.

Existem dois sistemas mais comuns de diagramação que ajudam na representação visual do modelo relacional: O diagrama de entidade-relacionamento DER, e o diagrama correlato IDEF utilizado no método IDEF1X criado pela Força aérea americana baseado no DER.

13 regras de Ted Codd

Os bancos de dados relacionais foram idealizados por Edgar Frank "Ted" Codd, um cientista de computação britânico que, enquanto trabalhava para a IBM, inventou o modelo relacional para a gestão de banco de dados, a base teórica para bancos de dados relacionais. Ele fez outras contribuições valiosas para a ciência da computação, mas o modelo relacional, uma teoria muito influente sobre gestão de dados gerais, continua sendo seu feito mais citado. Em 1970 ele apareceu com 13 leis (numeradas de 0 a 12) que descreveriam o que é um banco de dados relacional e o que é um Sistema Gerenciador de Banco de Dados Relacionais faz e, várias leis de normalização que descrevem as propriedades de dados relacionais. Apenas os dados que haviam sido normalizados poderiam ser considerados relacionais.

1. Regra Fundamental: Um SGBD relacional deve gerenciar seus dados usando apenas suas capacidades relacionais.

2. Regra da informação: Toda informação deve ser representada de uma única forma, como dados em uma tabela.
3. Regra da garantia de acesso: Todo dado (valor atômico) pode ser acessado logicamente (e unicamente) usando o nome da tabela, o valor da chave primária da linha e o nome da coluna.
4. Tratamento sistemático de valores nulos: Os valores nulos (diferente do zero, da string vazia, da string de caracteres em brancos e outros valores não nulos) existem para representar dados não existentes de forma sistemática e independente do tipo de dado.
5. Catálogo dinâmico on-line baseado no modelo relacional: A descrição do banco de dados é representada no nível lógico como dados comuns (isso é, em tabelas), permitindo que usuários autorizados apliquem as mesmas formas de manipular dados aplicados aos dados comuns ao consultá-los.
6. Regra da sub-linguagem compreensiva: Um sistema relacional pode suportar várias linguagens e formas de uso, porém deve possuir ao menos uma linguagem com sintaxe bem definida e expressa por cadeia de caracteres e com habilidade de apoiar a definição de dados, a definição de visões, a manipulação de dados, as restrições de integridade, a autorização e a fronteira de transações.
7. Regra da atualização de visões: Toda visão que for teoricamente atualizável será também atualizável pelo sistema.
8. Inserção, atualização e eliminação de alto nível: A capacidade de manipular a relação base ou relações derivadas como um operador único não se aplica apenas a recuperação de dados, mas também a inserção, alteração e eliminação de dados.
9. Independência dos dados físicos: Programas de aplicação ou atividades de terminal permanecem logicamente inalteradas quaisquer que sejam as modificações na representação de armazenagem ou métodos de acesso internos.
10. Independência lógica de dados: Programas de aplicação ou atividades de terminal permanecem logicamente inalteradas quaisquer que sejam as mudanças de informação que permitam teoricamente a não alteração das tabelas base.

11. Independência de integridade: As relações de integridade específicas de um banco de dados relacional devem ser definidas em uma sub-linguagem de dados e armazenadas no catálogo (e não em programas).
12. Independência de distribuição: A linguagem de manipulação de dados deve possibilitar que as aplicações permaneçam inalteradas estejam os dados centralizados ou distribuídos fisicamente.
13. Regra da Não-subversão: Se o sistema relacional possui uma linguagem de baixo nível (um registro por vez), não deve ser possível subverter ou ignorar as regras de integridade e restrições definidas no alto nível (muitos registros por vez).

Noções de conjunto

No estudo de Conjuntos, trabalhamos com alguns conceitos primitivos, que devem ser entendidos e aceitos sem definição. Para um estudo mais aprofundado sobre a Teoria dos Conjuntos, pode-se ler: *Naive Set Theory*, P.Halmos ou *Axiomatic Set Theory*, P.Suppes. O primeiro deles foi traduzido para o português sob o título (nada ingênuo de): Teoria *Ingênua* dos Conjuntos.

Origem

A teoria teve seu início com a publicação em 1874 de um trabalho de Cantor que tratava sobre a comparação de coleções infinitas. O trabalho apresentava uma forma de comparar conjuntos infinitos pelo "casamento" 1-1 entre os elementos destes conjuntos. Desde 1638, com Galileu Galilei, sabe-se que se pode obter uma correspondência 1-1 entre os números inteiros e seus quadrados, o que violava a concepção euclidiana de que o todo é sempre maior que qualquer uma de suas partes. Esta aplicação da correspondência 1-1 permitiu a Cantor introduzir um método de diagonalização, que por contradição, permitia provar que o conjunto dos números reais não tinha correspondência 1-1 com o conjunto dos números inteiros. Isto, mais tarde, levou ao desenvolvimento do conceito de contínuo por Richard Dedekind.

Iniciando com estas descobertas, Cantor acabou desenvolvendo uma teoria dos conjuntos abstratos, que constitui-se em uma generalização do conceito de conjunto.

Conjunto

Na teoria dos conjuntos, um conjunto é descrito como uma coleção de objetos bem definidos. Estes objetos são chamados de elementos ou membros do conjunto. Os objetos podem ser qualquer coisa: números, pessoas, outros conjuntos, etc. Por exemplo, 4 é um número do conjunto dos inteiros. Como pode ser visto por este exemplo, os conjuntos podem ter um número infinito de elementos.

Relações

Se x é um membro de A , então também é dito que x **pertence a** A , ou que x está em A . Neste caso, escrevemos $x \in A$. (O símbolo " \in " é derivado da letra grega épsilon, " ϵ ", introduzida por Giuseppe Peano em 1888). O símbolo \notin é às vezes usado para escrever $x \notin A$, ou " x não pertence a A ".

Dois conjuntos A e B são **iguais** quando possuem precisamente os mesmos elementos, isto é, se cada elemento de A é um elemento de B e cada elemento de B é um elemento de A . Um conjunto é completamente determinado por seus elementos; a descrição é imaterial. Por exemplo, o conjunto com os números 2, 3 e 5 é igual ao conjunto de todos os números primos menores que 6. Se A e B são iguais, então é representado simbolicamente por $A = B$ (como de costume). Também é permitido um **conjunto vazio**, muitas vezes representado pelo símbolo \emptyset : um conjunto sem elementos. Já que um conjunto é determinado completamente por seus elementos, pode haver apenas um conjunto vazio.

Se A é um subconjunto do conjunto B , diz-se que A está contido em B . Neste caso, escreve-se $A \subset B$. Já o símbolo $\not\subset$ é usado para escrever $A \not\subset B$, ou "o conjunto A não está contido no conjunto B ".

Alguns conceitos

Um conjunto representa uma coleção de objetos.

- O conjunto de todos os brasileiros.
- O conjunto de todos os números naturais.
- O conjunto de todos os números reais tal que $x^2 - 4 = 0$.

Em geral, um conjunto é denotado por uma letra maiúscula do alfabeto: A , B , C , ..., Z .

Elemento: é um dos componentes de um conjunto.

- José da Silva é um elemento do conjunto dos brasileiros.

- b. 1 é um elemento do conjunto dos números naturais.
- c. -2 é um elemento do conjunto dos números reais que satisfaz à equação $x^2-4=0$.

Em geral, um elemento de um conjunto, é denotado por uma letra minúscula do alfabeto: a, b, c, ..., z.

Pertinência: é a característica associada a um elemento que faz parte de um conjunto.

- a. José da Silva pertence ao conjunto dos brasileiros.
- b. 1 pertence ao conjunto dos números naturais.
- c. -2 pertence ao conjunto de números reais que satisfaz à equação $x^2-4=0$.

Símbolo de pertinência: Se um elemento pertence a um conjunto utilizamos o símbolo \in que se lê: "pertence".

Para afirmar que 1 é um número natural ou que 1 pertence ao conjunto dos números naturais, escrevemos:

$$1 \in \mathbb{N}$$

Para afirmar que 0 não é um número natural ou que 0 não pertence ao conjunto dos números naturais, escrevemos:

$$0 \notin \mathbb{N}$$

Um símbolo matemático muito usado para a negação é a barra / traçada sobre o símbolo normal.

Algumas notações para conjuntos

Muitas vezes, um conjunto é representado com os seus elementos dentro de duas chaves { e } através de duas formas básicas e de uma terceira forma geométrica:

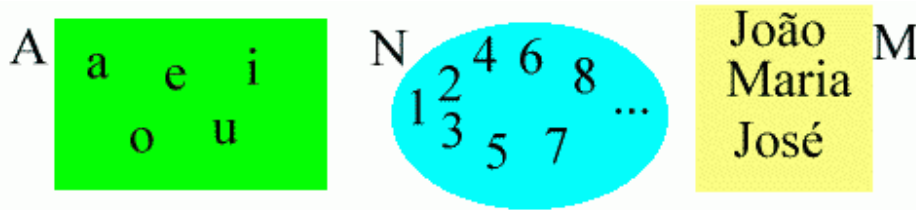
Apresentação: Os elementos do conjunto estão dentro de duas chaves { e }.

- a. $A=\{a,e,i,o,u\}$
- b. $N=\{1,2,3,4,\dots\}$
- c. $M=\{\text{João},\text{Maria},\text{José}\}$

Descrição: O conjunto é descrito por uma ou mais propriedades.

- a. $A=\{x: x \text{ é uma vogal}\}$
- b. $N=\{x: x \text{ é um número natural}\}$
- c. $M=\{x: x \text{ é uma pessoa da família de Maria}\}$

Diagrama de Venn-Euler: (lê-se: "Ven-óiler") Os conjuntos são mostrados graficamente.



Subconjuntos

Dados os conjuntos A e B, diz-se que A está contido em B, denotado por $A \subset B$, se todos os elementos de A também estão em B. Algumas vezes diremos que um conjunto A está propriamente contido em B, quando o conjunto B, além de conter os elementos de A, contém também outros elementos. O conjunto A é denominado *subconjunto* de B e o conjunto B é o *superconjunto* que contém A.

Alguns conjuntos especiais

Conjunto vazio: É um conjunto que não possui elementos. É representado por $\{ \}$ ou por \emptyset . O conjunto vazio está contido em todos os conjuntos.

Conjunto universo: É um conjunto que contém todos os elementos do contexto no qual estamos trabalhando e também contém todos os conjuntos desse contexto. O conjunto universo é representado por uma letra U. Na sequência não mais usaremos o conjunto universo.

Reunião de conjuntos

A reunião dos conjuntos A e B é o conjunto de todos os elementos que pertencem ao conjunto A **ou** ao conjunto B.

$$A \cup B = \{ x: x \in A \text{ ou } x \in B \}$$

Exemplo: Se $A = \{a, e, i, o\}$ e $B = \{3, 4\}$ então $A \cup B = \{a, e, i, o, 3, 4\}$.

Interseção de conjuntos

A interseção dos conjuntos A e B é o conjunto de todos os elementos que pertencem ao conjunto A **e** ao conjunto B.

$$A \cap B = \{ x: x \in A \text{ e } x \in B \}$$

Exemplo: Se $A=\{a,e,i,o,u\}$ e $B=\{1,2,3,4\}$ então $A \cap B = \emptyset$.



Quando a interseção de dois conjuntos A e B é o conjunto vazio, dizemos que estes conjuntos são **disjuntos**.

Propriedades dos conjuntos

Fechamento: Quaisquer que sejam os conjuntos A e B, a reunião de A e B, denotada por $A \cup B$ e a interseção de A e B, denotada por $A \cap B$, ainda são conjuntos no universo.

1. **Reflexiva:** Qualquer que seja o conjunto A, tem-se que:

$$A \cup A = A \quad \text{e} \quad A \cap A = A$$

2. **Inclusão:** Quaisquer que sejam os conjuntos A e B, tem-se que:

$$A \subset A \cup B, \quad B \subset A \cup B, \quad A \cap B \subset A, \quad A \cap B \subset B$$

3. **Inclusão relacionada:** Quaisquer que sejam os conjuntos A e B, tem-se que:

$$\begin{aligned} A \subset B &\text{ equivale a } A \cup B = B \\ A \subset B &\text{ equivale a } A \cap B = A \end{aligned}$$

4. **Associativa:** Quaisquer que sejam os conjuntos A, B e C, tem-se que:

$$\begin{aligned} A \cup (B \cap C) &= (A \cup B) \cap C \\ A \cap (B \cup C) &= (A \cap B) \cup C \end{aligned}$$

5. **Comutativa:** Quaisquer que sejam os conjuntos A e B, tem-se que:

$$\begin{aligned} A \cup B &= B \cup A \\ A \cap B &= B \cap A \end{aligned}$$

6. **Elemento neutro para a reunião:** O conjunto vazio \emptyset é o elemento neutro para a reunião de conjuntos, tal que para todo conjunto A, se tem:

$$A \cup \emptyset = A$$

7. **Elemento "nulo" para a interseção:** A interseção do conjunto vazio \emptyset com qualquer outro conjunto A, fornece o próprio conjunto vazio.

$$A \cap \emptyset = \emptyset$$

8. **Elemento neutro para a interseção:** O conjunto universo U é o elemento neutro para a interseção de conjuntos, tal que para todo conjunto A, se tem:

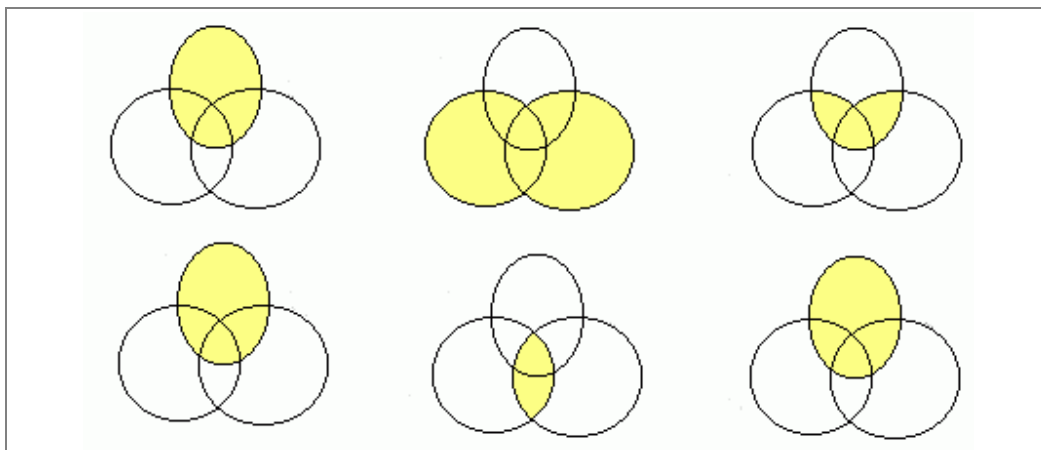
$$A \cap U = A$$

9. **Distributiva:** Quaisquer que sejam os conjuntos A, B e C, tem-se que:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

Os gráficos abaixo mostram a distributividade.

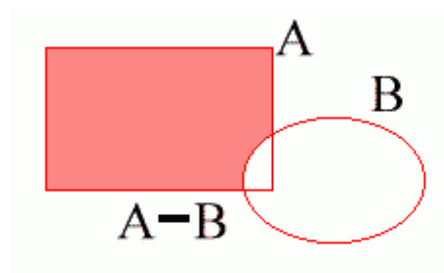


Diferença de conjuntos

A diferença entre os conjuntos A e B é o conjunto de todos os elementos que pertencem ao conjunto A e *não* pertencem ao conjunto B.

$$A - B = \{x: x \in A \text{ e } x \notin B\}$$

Do ponto de vista gráfico, a diferença pode ser vista como:



Complemento de um conjunto

O complemento do conjunto B contido no conjunto A, denotado por $C_A B$, é a diferença entre os conjuntos A e B, ou seja, é o conjunto de todos os elementos que pertencem ao conjunto A e *não* pertencem ao conjunto B.

$$C_A B = A - B = \{x: x \in A \text{ e } x \notin B\}$$

Graficamente, o complemento do conjunto B no conjunto A, é dado por:



Quando não há dúvida sobre o universo U em que estamos trabalhando, simplesmente utilizamos a letra **c** posta como expoente no conjunto, para indicar o complemento deste conjunto. Muitas vezes usamos a palavra **complementar** no lugar de complemento.

Exemplos: $\emptyset^c = U$ e $U^c = \emptyset$.

Leis de Augustus De Morgan

1. O complementar da reunião de dois conjuntos A e B é a interseção dos complementares desses conjuntos.

$$(A \cup B)^c = A^c \cap B^c$$

2. O complementar da reunião de uma coleção finita de conjuntos é a interseção dos complementares desses conjuntos.

$$(A_1 \cup A_2 \cup \dots \cup A_n)^c = A_1^c \cap A_2^c \cap \dots \cap A_n^c$$

3. O complementar da interseção de dois conjuntos A e B é a reunião dos complementares desses conjuntos.

$$(A \cap B)^c = A^c \cup B^c$$

4. O complementar da interseção de uma coleção finita de conjuntos é a reunião dos complementares desses conjuntos.

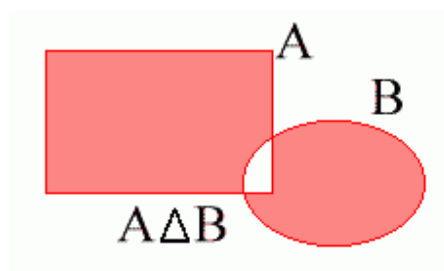
$$(A_1 \cap A_2 \cap \dots \cap A_n)^c = A_1^c \cup A_2^c \cup \dots \cup A_n^c$$

Diferença simétrica

A diferença simétrica entre os conjuntos A e B é o conjunto de todos os elementos que pertencem à reunião dos conjuntos A e B e *não* pertencem à interseção dos conjuntos A e B.

$$A \Delta B = \{ x: x \in A \cup B \text{ e } x \notin A \cap B \}$$

O diagrama de Venn-Euler para a diferença simétrica é:



A Linguagem SQL

Introdução ao SQL

Structured Query Language, ou Linguagem de Consulta Estruturada ou SQL, é uma linguagem de pesquisa declarativa para banco de dados relacional (base de dados relacional). Muitas das características originais do SQL foram inspiradas na álgebra relacional. O SQL foi desenvolvido originalmente no início dos anos 70 nos laboratórios da IBM em San Jose, dentro do projeto System R, que tinha por objetivo demonstrar a viabilidade da implementação do modelo relacional proposto por E. F. Codd. O nome original da linguagem era SEQUEL, acrônimo para "Structured English Query Language" (Linguagem de Consulta Estruturada em Inglês), vindo daí o fato de, até hoje, a sigla, em inglês, ser comumente pronunciada "síquel" ao invés de "ês-kiú-él", letra a letra. No entanto, em português, a pronúncia mais corrente é a letra a letra: "ésse-quê-éle". A linguagem SQL é um grande padrão de banco de dados. Isto decorre da sua simplicidade e facilidade de uso. Ela se diferencia de outras linguagens de consulta a banco de dados no sentido em que uma consulta SQL especifica a forma do resultado e não o caminho para chegar a ele. Ela é uma linguagem declarativa em oposição a outras linguagens procedurais. Isto reduz o ciclo de aprendizado daqueles que se iniciam na linguagem. Embora o SQL tenha sido originalmente criado pela IBM, rapidamente surgiram vários "dialectos" desenvolvidos por outros produtores. Essa expansão levou à necessidade de ser criado e adaptado um padrão para a linguagem. Esta tarefa foi realizada pela American National Standards Institute (ANSI) em 1986 e ISO em 1987. O SQL foi revisto em 1992 e a esta versão foi dado o nome de SQL-92. Foi revisto novamente em 1999 e 2003 para se tornar SQL:1999 (SQL3) e SQL:2003, respectivamente. O SQL:1999 usa expressões regulares de emparelhamento, queries recursivas e gatilhos (triggers). Também foi feita uma adição controversa de tipos não-escalados e algumas características de orientação a objeto. O SQL:2003 introduz características relacionadas ao XML, sequências padronizadas e colunas com valores de auto-generalização (inclusive colunas-identidade). Tal como dito anteriormente, o SQL, embora padronizado pela ANSI e ISO, possui muitas variações e extensões produzidos pelos diferentes fabricantes de sistemas gerenciadores de bases de dados. Tipicamente a linguagem pode ser migrada de plataforma para plataforma sem mudanças estruturais principais. Outra aproximação é permitir para código de idioma procedural ser embutido e interagir com o banco de dados. Por exemplo, o Oracle e outros incluem Java na base de dados, enquanto o PostgreSQL permite que funções sejam escritas em Perl, Tcl, ou C, entre outras linguagens.

Características do SQL

SQL é uma linguagem com o Inglês. Ela usa palavras semelhantes como select, insert, delete e também parte de conjunto de comandos. SQL é uma linguagem não procedural: você especifica qual informação você quer; não como você quer pegá-la. Em outras palavras, o SQL não requer que você especifique o método de acesso para os dados. Todas as declarações de pesquisas são otimizadas - uma parte do RDBMS - para determinar a fantástico método de retirar os especificados dados Esta característica faz dele o mais fácil para você concentrar-se e obter o resultado desejado. SQL processa

conjuntos de registros melhor do que um único registro no tempo. A melhor forma comum de conjunto de registros é uma tabela. SQL pode ser usado por uma faixa de usuários incluindo DBAs, Programadores, Gerentes, e muitos outros tipos de usuários finais. SQL provém comandos para uma variedade de tarefas incluindo:

- pesquisando dados
- inserindo, alterando apagando linhas em uma tabela
- criando, modificando, e apagando objetos do Banco de Dados
- controlando acesso para Banco de Dados e objetos do Banco de Dados

garantindo a consistência do banco de dados

Os mais fáceis Sistemas Gerenciadores de Banco de Dados geralmente usam uma linguagem separada para cada categoria acima. SQL unificou todas essas tarefas em uma única linguagem. SQL tem tornado de fato a linguagem industrial padrão para os Bancos de Dados Relacionais. O Instituto Nacional Padrão Americano (ANSI) adotou o SQL como linguagem padrão para RDBMS em 1986. A Organização de Padrões Internacionais (ISO) tem adotado também o SQL como linguagem padrão para o RDBMS. Todos os principais RDBMSs usam o SQL e os outros vendedores de RDBMS pretendem completar com o padrão ANSI.

O Conjunto de Comandos SQL

| Comandos | Descrições |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SELECT | Esse é o mais comum comando usado, ele é usado para retirar dados de uma Tabela |
| INSERT UPDATE DELETE | Esses três comandos são usados para o preenchimento de novas linhas, modificando linhas existentes e removendo linhas não desejadas das tabelas nos Banco de Dados respectivos. (Eles são algumas vezes coletivamente conhecidos como DML ou comandos de Linguagem de Manipulação dos Dados). |
| CREATE ALTER DROP | Esses três comandos são usados dinamicamente para configurar, modificam e removem várias estruturas de dados como: tabelas, visões, índices. (Eles são algumas vezes conhecidos como DDL ou comandos de Linguagem de Definição de Dados). |
| GRANT REVOKE | Esses dois comandos são usados para dar ou remover direitos de acesso para ambos Banco de Dados ORACLE e nas Estruturas dentro dele.. |

Escrevendo Comandos SQL

Quando se escreve comandos SQL, é importante lembrar das seguintes regras e diretrizes de modo que a construção seja válida e fáceis para ler e editar:

- Os comandos SQL podem ser sobre uma ou mais linhas.
- Clausulas são usualmente localizadas em linhas separadas.
- Tabulações podem ser usadas
- Palavras comandos não podem ser divididas transversalmente nas linhas
- Os comandos SQL não podem ser exemplos sensitivos (a não ser indicando diferença)
- Um comando SQL é entrado no SQL pronto, e subseqüentemente as linhas são numeradas.,isto é chamado de SQL buffer.
- Geralmente uma declaração pode estar corrente várias vezes no buffer, e ela pode ser executada de várias formas:
 - colocado um ponto e vírgula(;) no final da última clausula.
 - Colocar um ; ou / no final da linha no buffer
 - colocar um / no SQL pronto
 - colocar um comando R(UN) no SQL pronto

O comando Select

A declaração SELECT retira informações do Banco de Dados, implementando todos os operadores da Álgebra Relacional. Neste simples forma, ele deve incluir:

1. Uma clausula SELECT, o qual terá a lista das colunas a serem mostradas. Ele é essencialmente um PROJECTION.
2. Uma clausula FROM, a qual especifica a tabela envolvida.

Várias declarações são válidas:

```
SELECT * FROM EMP;
```

```
SELECT  
*  
FROM  
EMP  
;
```

```
SELECT *  
FROM EMP;
```

Par listar todos os números de departamentos, nome dos empregados e números dos gerentes na tabela EMP você entra com o seguinte comando:

```
SELECT DEPTNO, ENAME, MGR
FROM   EMP;
```

| DEPTNO | ENAME | MGR |
|--------|--------|------|
| 20 | SMITH | 7902 |
| 30 | ALLEN | 7698 |
| 30 | WARD | 7698 |
| 20 | JONES | 7839 |
| 30 | MARTIN | 7698 |
| 30 | BLAKE | 7839 |
| 10 | CLARK | 7839 |
| 20 | SCOTT | 7566 |
| 10 | KING | |
| 30 | TURNER | 7698 |
| 20 | ADAMS | 7788 |
| 30 | JAMES | 7698 |
| 20 | FORD | 7566 |
| 10 | MILLER | 7782 |

Note que os nomes de colunas estão separados por uma vírgula.

Ele possibilita selecionar todas as colunas da tabela, especificando um * (asterisco) depois do SELECT palavra comando.

```
SELECT * FROM EMP;
```

| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|-------|--------|-----------|------|-----------|----------|----------|--------|
| 7369 | SMITH | CLERK | 7902 | 13-JUN-83 | 800.00 | | 20 |
| 7499 | ALLEN | SALESMAN | 7698 | 15-AUG-83 | 1,600.00 | 300.00 | 30 |
| 7521 | WARD | SALESMAN | 7698 | 26-MAR-84 | 1,250.00 | 500.00 | 30 |
| 7566 | JONES | MANAGER | 7839 | 31-OCT-83 | 2,975.00 | | 20 |
| 7654 | MARTIN | SALESMAN | 7698 | 05-DEC-83 | 1,250.00 | 1,400.00 | 30 |
| 7698 | BLAKE | MANAGER | 7839 | 11-JUN-84 | 2,850.00 | | 30 |
| 7782 | CLARK | MANAGER | 7839 | 14-MAY-84 | 2,450.00 | | 10 |
| 7788 | SCOTT | ANALYST | 7566 | 05-MAR-84 | 3,000.00 | | 20 |
| 7839 | KING | PRESIDENT | | 09-JUL-84 | 5,000.00 | | 10 |
| 7844 | TURNER | SALESMAN | 7698 | 04-JUN-84 | 1,500.00 | .00 | 30 |
| 7876 | ADAMS | CLERK | 7788 | 04-JUN-84 | 1,100.00 | | 20 |
| 7900 | JAMES | CLERK | 7698 | 23-JUL-84 | 950.00 | | 30 |
| 7902 | FORD | ANALYST | 7566 | 05-DEC-83 | 3,000.00 | | 20 |
| 7934 | MILLER | CLERK | 7782 | 21-NOV-83 | 1,300.00 | | 10 |

Outros itens da cláusula SELECT

É possível incluir outros itens na cláusula SELECT.

- Expressões Aritméticas

- Colunas sinônimas
- Concatenação de colunas
- Literais

Todas estas opções permitem ao usuário pesquisar os dados e manipulá-los para as pesquisas propostas; por exemplo, fazendo cálculos, juntando colunas uma com a outra, ou mostrando pedaços de textos.

Expressões Aritméticas

Uma expressão é a combinação de um ou mais valores, operadores, e funções os quais avaliam para um valor. Expressões Aritméticas podem conter nome de colunas, valores numéricos constantes e operadores aritméticos:

| Operadores | Descrições |
|------------|---------------|
| + | Adição |
| - | Subtração |
| * | Multiplicação |
| / | Divisão |

```
SELECT ENAME, SAL*12, COMM FROM EMP;
```

```
EMPNO      SAL*12      COMM
```

```
-----
```

| | | |
|------|-------|----------|
| 7369 | 9600 | |
| 7499 | 19200 | 300.00 |
| 7521 | 15000 | 500.00 |
| 7566 | 35700 | |
| 7654 | 15000 | 1,400.00 |
| 7698 | 34200 | |
| 7782 | 29400 | |
| 7788 | 36000 | |
| 7839 | 60000 | |
| 7844 | 18000 | .00 |
| 7876 | 13200 | |
| 7900 | 11400 | |
| 7902 | 36000 | |
| 7934 | 15600 | |

Se a sua expressão aritmética conter mais de um operador a prioridade é *, / primeiro, então +, - segundo (deixe para direita se existir operadores com a mesma prioridade).

No exemplo seguinte a multiplicação (250*12) é avaliada primeiro; então o valor do salário é adicionado no resultado da multiplicação (3000). Somente para linha do Smith: 800+3000=3800

```
select ename, sal + 250 * 12
from emp;
```

| ENAME | SAL+250*12 |
|-------|------------|
| SMITH | 3800 |
| ALLEN | 4600 |
| WARD | 4250 |
| JONES | 5975 |

Parênteses podem ser usados para especificar a ordem na qual serão executados os operadores, se, por exemplo, a adição é requerida antes da multiplicação.

```
select ename, (sal + 250) * 12
from emp;
```

| ENAME | (SAL+250)*12 |
|-------|--------------|
| SMITH | 12600 |
| ALLEN | 22200 |
| WARD | 18000 |
| JONES | 38700 |

Colunas Sinônimas

Quando mostramos o resultado de uma pesquisa, o SQL*Plus por exemplo normalmente é mostrado o nome as colunas selecionadas como cabeçalho. Em alguns exemplos ele pode ser sem sentido. Você pode modificar o cabeçalho de uma coluna usando sinônimos(alias). Uma coluna sinônima é um cabeçalho de coluna alternativo na saída. Especifique o sinônimo (alias) depois da coluna na lista do SELECT. O cabeçalho sinônimo padrão será fornecido sem espaços em branco, amenos que o sinônimo esteja dentro de aspas duplos (“ ”). Para mostrar um cabeçalho ANNSAL para o salário anual em vez de SAL*12 usamos uma coluna sinônima.

```
SELECT ENAME, SAL*12 ANNSAL, COMM
FROM EMP;
```

| EMPNO | ANNSAL | COMM |
|-------|--------|----------|
| 7369 | 9600 | |
| 7499 | 19200 | 300.00 |
| 7521 | 15000 | 500.00 |
| 7566 | 35700 | |
| 7654 | 15000 | 1,400.00 |
| 7698 | 34200 | |
| 7782 | 29400 | |
| 7788 | 36000 | |
| 7839 | 60000 | |
| 7844 | 18000 | .00 |
| 7876 | 13200 | |
| 7900 | 11400 | |
| 7902 | 36000 | |
| 7934 | 15600 | |

Nota: A declaração de colunas sinônimas no SQL, podem ser usadas unicamente na clausula SELECT.

O Operador de Concatenação

O Operador de Concatenação (||) permite que as colunas sejam juntadas com outras colunas, expressões aritméticas ou valores constantes para criar uma expressão alfanumérica. Colunas ficam lado a lado com operadores para formarem uma única coluna.

Para combinar EMPNO e ENAME e obter o sinônimo EMPLOYEE, entre-se:

```
SELECT EMPNO || ENAME EMPLOYEE FROM EMP;
```

```
EMPLOYEE
```

```
-----
7369SMITH
7499ALLEN
7521WARD
7566JONES
7654MARTIN
7698BLAKE
7782CLARK
7788SCOTT
7839KING
7844TURNER
7876ADAMS
7900JAMES
7902FORD
7934MILLER
```

Literais

Um literal é um ou mais caracteres, expressões, números incluídos na lista do SELECT o qual não é um nome de coluna ou de um sinônimo. Um literal na lista do SELECT terá uma saída para cada linha retornada. Literais de livre formatos de textos podem ser incluídos no resultado da pesquisa, e são tratados como uma coluna na lista do SELECT. Datas e caracteres alfanuméricos devem ser colocados entre aspas simples('); números não precisam de aspas simples.

As declarações seguintes contêm literais selecionados com concatenação e colunas sinônimas.

```
SELECT      EMPNO || '-' || ENAME EMPLOYEE,
            'WORKS IN DEPARTMENT',
            DEPTNO
FROM        EMP;
```

```
EMPLOYEE      'WORKSINDEPARTAMENT' DEPTNO
-----
7369-SMITH    WORKS IN DEPARTMENT      20
7499-ALLEN    WORKS IN DEPARTMENT      30
```

| | | |
|------------|---------------------|----|
| 7521-WARD | WORKS IN DEPARTMENT | 30 |
| 7566-JONES | WORKS IN DEPARTMENT | 20 |

Manuseando Valores Nulos

Se uma linha necessitar de dados para uma coluna particular, se diz que a coluna está nula. Um valor nulo é um valor indisponível e desconhecido. O valor nulo não é zero. Zero é um número. Valores Nulos são manuseados corretamente pelo SQL. Se algum valor de uma coluna em uma expressão for nulo, o resultado será nulo.. Na declaração seguinte, salesmen tem um resultado na remuneração:

```
SELECT ENAME, SAL*12+COMM ANNUAL_SAL FROM EMP;
```

| ENAME | ANNUAL_SAL |
|--------|------------|
| ----- | ----- |
| SMITH | |
| ALLEN | 19500 |
| WARD | 15500 |
| JONES | |
| MARTIN | 16400 |
| BLAKE | |
| CLARK | |
| SCOTT | |
| KING | |
| TURNER | 18000 |
| ADAMS | |
| JAMES | |
| FORD | |
| MILLER | |

Função NVL

Na ordem para realizar o resultado para todos os empregados, é necessário converter os valores nulos para numéricos. Nós usamos a função NVL para converter valores nulos para não nulos. Usando a Função NVL para converter valores nulos da declaração anterior para zero.

```
SELECT ENAME, SAL*12NVL (COMM, 0) ANNUL_SAL FROM EMP;
```

| ENAME | ANNUAL_SAL |
|--------|------------|
| ----- | ----- |
| SMITH | 9600 |
| ALLEN | 19500 |
| WARD | 15500 |
| JONES | 35700 |
| MARTIN | 16400 |
| BLAKE | 34200 |
| CLARK | 29400 |
| SCOTT | 36000 |
| KING | 60000 |
| TURNER | 18000 |
| ADAMS | 13200 |
| JAMES | 11400 |
| FORD | 36000 |
| MILLER | 15600 |

A função NVL conta com dois argumentos:

1. uma expressão
2. um valor não nulo.

Note que você pode usar a função NVL para converter qualquer tipo de valor nulo.

NVL(COLUNA_DATA_NULA,'22/01/2011')

NVL(COLUNA_NUMÉRICA_NULA,21)

NVL(COLUNA_ALFANUMÉRICA_NULA,'QUALQUER VALOR')

Prevenindo a Seleção de Linhas Duplicadas

A menos que você indique de outra maneira, SQL*Plus mostrará os resultados da pesquisa sem eliminar as duplicações. Para listar todos os números de departamentos da tabela EMP, faça:

```
SELECT DEPTNO FROM EMP;
```

DEPTNO

20

30

30

20

30

30

10

20

10

30

20

30

20

10

A cláusula DISTINCT

Para eliminar valores duplicados no resultado, incluímos o DISTINCT qualificador no comando SELECT.

Para eliminar os valores Duplicados mostrados no exemplo anterior, faça:

```
SELECT DISTINCT DEPTNO FROM EMP;
```

DEPTNO

10

20

30

Várias colunas podem ser especificadas depois do qualificador DISTINCT a palavra DISTINCT agirá sobre todas as colunas selecionadas.

Para mostrar distintos valores de DEPTNO e JOB, faça:

```
SELECT DISTINCT DEPTNO, JOB FROM EMP;
```

DEPTNO JOB

```

10 CLERK
10 MANAGER
10 PRESIDENT
20 ANALYST
20 CLERK
20 MANAGER
30 CLERK
30 MANAGER
30 SALESMAN

```

Essa mostra a lista de todas as combinações de diferentes empregos e números de departamentos. Note que o DISTINCT qualificador pode unicamente referir para uma vez e precisa do comando SELECT.

A cláusula ORDER BY

Normalmente a ordem das linhas retornadas de uma pesquisa é indefinida. A cláusula ORDER BY pode ser usada para ordenar as linhas. Se usado, o ORDER BY precisa sempre ser a última cláusula da declaração SELECT.

Para ordenar pelo ENAME, faça:

```
SELECT ENAME, JOB, SAL*12, DEPTNO FROM EMP ORDER BY ENAME;
```

| ENAME | JOB | SAL*12 | DEPTNO |
|--------|-----------|--------|--------|
| ----- | ----- | ----- | ----- |
| ADAMS | CLERK | 13200 | 20 |
| ALLEN | SALESMAN | 19500 | 30 |
| BLAKE | MANAGER | 34200 | 30 |
| CLARK | MANAGER | 29400 | 10 |
| FORD | ANALYST | 36000 | 20 |
| JAMES | CLERK | 11400 | 30 |
| JONES | MANAGER | 35700 | 20 |
| KING | PRESIDENT | 60000 | 10 |
| MARTIN | SALESMAN | 16400 | 30 |
| MILLER | CLERK | 15600 | 10 |
| SCOTT | ANALYST | 36000 | 20 |
| SMITH | CLERK | 9600 | 20 |
| TURNER | SALESMAN | 18000 | 30 |
| WARD | SALESMAN | 15500 | 30 |

Padrão da Ordenação dos Dados

O padrão da ordem de ordenação é ascendente.

- valores numéricos infinitos primeiro
- valores de data primeiro
- valores alfanuméricos

Invertendo o padrão de ordenação

Para inverter essa ordem, o comando DESC(Decrescente)do depois do nome das colunas da cláusula ORDER BY.

Para inverter a ordem da coluna HIREDATE, somente aquela data mais recente estará mostrada primeiro, faça:

```
SELECT ENAME, JOB, HIREDATE FROM EMP ORDER BY HIREDATE DESC;
```

| ENAME | JOB | HIREDATE |
|--------|-----------|-----------|
| ----- | ----- | ----- |
| JAMES | CLERK | 23-JUL-84 |
| KING | PRESIDENT | 09-JUL-84 |
| BLAKE | MANAGER | 11-JUN-84 |
| ADAMS | CLERK | 04-JUN-84 |
| TURNER | SALESMAN | 04-JUN-84 |
| CLARK | MANAGER | 14-MAY-84 |
| WARD | SALESMAN | 26-MAR-84 |
| SCOTT | ANALYST | 05-MAR-84 |
| MARTIN | SALESMAN | 05-DEC-83 |
| FORD | ANALYST | 05-DEC-83 |
| MILLER | CLERK | 21-NOV-83 |
| JONES | MANAGER | 31-OCT-83 |
| ALLEN | SALESMAN | 15-AUG-83 |
| SMITH | CLERK | 13-JUN-83 |

Ordenação por várias colunas

É possível na cláusula ORDER BY usar mais de uma coluna. O limite de colunas é de fato o número de colunas da tabela. Na cláusula ORDER BY especifica-se as colunas pelo que as linhas serão ordenadas, separando as por vírgula. Se algumas ou todas são invertidas especifique DESC depois de alguma ou cada uma das colunas.

Para ordenar por duas colunas, e mostrar inversa a ordem do salário, faça:

```
SELECT DEPTNO, JOB, ENAME
FROM EMP
ORDER BY DEPTNO, SAL DESC;
```

| DEPTNO | JOB | ENAME |
|--------|-----------|--------|
| ----- | ----- | ----- |
| 10 | PRESIDENT | KING |
| 10 | MANAGER | CLARK |
| 10 | CLERK | MILLER |
| 20 | ANALYST | SCOTT |
| 20 | ANALYST | FORD |
| 20 | MANAGER | JONES |
| 20 | CLERK | ADAMS |
| 20 | CLERK | SMITH |
| 30 | MANAGER | BLAKE |
| 30 | SALESMAN | ALLEN |
| 30 | SALESMAN | TURNER |
| 30 | SALESMAN | WARD |
| 30 | SALESMAN | MARTIN |
| 30 | CLERK | JAMES |

Para ordenar por uma coluna, ela não precisa necessariamente estar declarada no SELECT.

AVISO:

A cláusula ORDER BY é usada na pesquisa quando você quer mostrar as linhas em uma ordem específica. Sem a cláusula ORDER BY as linhas são retornadas na ordem conveniente para o ORACLE, e você não deve contar com essa ordem nas próximas

pesquisas. O comando não altera a ordem dos dados que estão armazenados no Banco de Dados ORACLE.

A Clausula WHERE

A clausula WHERE corresponde aos Operadores de Restrições da Álgebra Relacional. Ele contém condições nas quais a linha precisa se encontrar em ordem para ser mostrada.

Estrutura da seleção com restrições.

```
SELECT      coluna(s)
FROM        tabela(s)
WHERE       certa condição a ser encontrada
```

A clausula WHERE pode comparar valores em uma coluna, valores literais, expressões aritméticas ou funções. A clausula WHERE conta com três elementos.

- Um nome de coluna
- Um operador de comparação
- Um nome de coluna, um constante, ou lista de valores.

Operadores de Comparação são usados na clausula WHERE e podem ser divididos em duas categorias, Lógicos e SQL.

Operadores Lógicos

Esses operadores lógicos testam as seguintes condições:

| Operador | Significado |
|----------|-----------------|
| = | igual a |
| > | maior que |
| >= | maior e igual a |
| < | menor que |
| <= | menor e igual a |

Alfanuméricos e Datas na clausula WHERE.

Colunas ORACLE devem ser: caracteres, numéricas ou data.

Alfanuméricos e Datas na clausula WHERE devem estar entre aspas simples. Os caracteres devem combinar com o valor da coluna a menos que seja modificado por funções. Referência para “Funções de Alfanuméricos” na Unidade 4.

Para listar os nomes, números, emprego e departamentos de todos os escriturários(CLERK).

```
SELECT ENAME, EMPNO, JOB, DEPTNO
FROM EMP
WHERE JOB = 'CLERK';
```

| ENAME | EMPNO | JOB | DEPTNO |
|--------|-------|-------|--------|
| SMITH | 7369 | CLERK | 20 |
| ADAMS | 7876 | CLERK | 20 |
| JAMES | 7900 | CLERK | 30 |
| MILLER | 7934 | CLERK | 10 |

Para encontrar todos os nomes de departamentos com número de departamento maior que 20, faça:

```
SELECT DNAME, DEPTNO
FROM DEPT
WHERE DEPTNO > 20;
```

| DNAME | DEPTNO |
|------------|--------|
| SALES | 30 |
| OPERATIONS | 40 |

Comparando uma coluna com outra coluna na mesma linha

Você pode comparar uma coluna com outra coluna na mesma linha, da mesma forma com um valor constante. Por exemplo, suponhamos que você quer encontrar os empregados os quais a comissão está maior que seu salário, faça:

```
SELECT ENAME, SAL, COMM
FROM EMP
WHERE COMM>SAL;
```

| ENAMES | SAL | COMM |
|--------|----------|----------|
| MARTIN | 1,250.00 | 1,400.00 |

Operadores SQL

Existem quatro operadores SQL os quais opera, com todos tipos de dados:

| Operador | Significado |
|---------------------|-----------------------------------|
| BETWEEN ... AND ... | Entre dois valores (inclusive) |
| IN(Lista) | compara uma lista de valores |
| LIKE | Compara um parâmetro alfanumérico |
| IS NULL | é um valor nulo |

O Operador BETWEEN

Testa uma faixa de valores, e inclusive do menor a maior faixa.

Suponhamos que nós quisemos ver aqueles empregados os quais o salário está entre 1000 e 2000:

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL BETWEEN 1000 AND 2000;
```

| ENAME | SAL |
|--------|----------|
| ALLEN | 1,600.00 |
| WARD | 1,250.00 |
| MARTIN | 1,250.00 |
| TURNER | 1,500.00 |
| ADAMS | 1,100.00 |
| MILLER | 1,300.00 |

Note aqueles valores especificados também compõem o resultado, e o menor precisa ser especificado primeiro.

O Operador IN

Testa os valores especificados em uma lista.

Para encontrar empregados que tenham um dos três números de MGR, faça:

```
SELECT EMPNO, ENAME, SAL, MGR
FROM EMP
WHERE MGR IN (7902,7566,7788)
```

| EMPNO | ENAME | SAL | MGR |
|-------|-------|----------|------|
| 7369 | SMITH | 800.00 | 7902 |
| 7788 | SCOTT | 3,000.00 | 7566 |
| 7876 | ADAMS | 1,100.00 | 7788 |
| 7902 | FORD | 3,000.00 | 7566 |

Se alfanuméricos ou datas forem usados na lista precisam ser colocados entre aspas simples(' ').

O Operador LIKE

Algumas vezes você precisa procurar valores que você não conhece exatamente. Usando o operador LIKE é possível selecionar linhas combinando parâmetros alfanuméricos. Dois símbolos podem ser usados para construir uma linha de procura.

| Símbolo | Representa |
|---------|----------------------------------------------|
| % | Várias seqüências de zero ou mais caracteres |
| _ | um número desejado de caracteres |

Para listar todos os empregados os quais o nome começa com a letra S, faça:

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE 'S%';
```

```
ENAME
-----
SMITH
SCOTT
```

Eles podem ser usados para encontrar um determinado número de caracteres. Por exemplo para listar todos empregados que tenham exatamente quatro caracteres de tamanho do nome.

```
SELECT ENAME
FROM EMP
WHERE ENAME LIKE '____';
```

```
ENAME
-----
WARD
KING
FORD
```

O % e o _ pode ser usado em várias combinações com literais alfanuméricos.

Operador IS NULL

Unicamente encontrar todos os empregados que não tenham gerente, você testará um valor nulo:

```
SELECT ENAME, MGR
FROM EMP
WHERE MGR IS NULL;
```

```
ENAME          MGR
-----
KING
```

Expressões Negativas

Os operadores seguintes são testes de negação:

| Operador | Descrição |
|----------|------------------------------|
| | |
| != | não igual para (VAX,UNIX,PC) |
| | |
| ^= | não igual para (IBM) |
| | |

| | |
|------------------|----------------------------------------------|
| <> | não igual para (todos sistemas operacionais) |
| NOT COLUNA_NOME= | não igual que |
| NOT COLUNA_NOME> | não maior que |

Operadores SQL

| Operador | Descrição |
|-------------|-------------------------------------------|
| NOT BETWEEN | tudo que estiver fora da faixa |
| NOT IN | tudo que não estiver na lista |
| NOT LIKE | tudo que não conter a linha de caracteres |
| IS NOT NULL | tudo que não for nulo |

Para encontrar empregados os quais o salário estiver fora da faixa, faça:

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL NOT BETWEEN 1000 AND 2000;
```

```
ENAME          SAL
-----
SMITH           800.00
JONES          2,975.00
BLAKE          2,850.00
CLARK           2,450.00
SCOTT           3,000.00
KING            5,000.00
JAMES           950.00
FORD            3,000.00
```

Para encontrar aqueles empregados os quais o cargo não comece com a letra M, faça:

```
SELECT ENAME, JOB
FROM EMP
WHERE JOB NOT LIKE 'M%';
```

```
ENAME          JOB
-----
SMITH          CLERK
ALLEN          SALESMAN
WARD           SALESMAN
MARTIN         SALESMAN
SCOTT          ANALYST
KING           PRESIDENT
TURNER         SALESMAN
ADAMS          CLERK
JAMES          CLERK
FORD           ANALYST
MILLER         CLERK
```


Para encontrar todos os empregados que tenham um gerente (MGR), faça:

```
SELECT ENAME, MGR
FROM EMP
WHERE MGR IS NOT NULL;
```

| ENAME | MGR |
|--------|------|
| SMITH | 7902 |
| ALLEN | 7698 |
| WARD | 7698 |
| JONES | 7839 |
| MARTIN | 7698 |
| BLAKE | 7839 |
| CLARK | 7839 |
| SCOTT | 7566 |
| TURNER | 7698 |
| ADAMS | 7788 |
| JAMES | 7698 |
| FORD | 7566 |
| MILLER | 7782 |

Nota:

Se um valor nulo é usado em uma comparação, então o operador de comparação deve ser IS ou IS NOT NULL. Se esses operadores não forem usados e valores nulos forem comparados, o resultado será sempre FALSO. Por exemplo, COMM <> NULL será sempre FALSO. O resultado será falso porque um valor nulo não pode ser igual ou não igual a qualquer outro valor, note que aquele erro não é elevado, o resultado é sempre falso.

Pesquisando Dados com Múltiplas Condições

Os operadores AND e OR devem ser usados para fazer composições de expressões lógicas. O predicado AND esperará que ambas as condições sejam verdadeiras; onde o predicado OR esperará uma das condições seja verdadeira. Nos dois seguintes exemplos as condições são as mesmas, o predicado é diferente. Veja como o resultado é dramaticamente alterado.

Para encontrar todos os escriturários que ganhem entre 1000 e 2000, faça:

```
SELECT EMPNO, ENAME, JOB, SAL
FROM EMP
WHERE SAL BETWEEN 1000 AND 2000
AND JOB = 'CLERK';
```

| EMPNO | ENAME | JOB | SAL |
|-------|--------|-------|----------|
| 7876 | ADAMS | CLERK | 1,100.00 |
| 7934 | MILLER | CLERK | 1,300.00 |

Para encontrar todos os empregados que são escriturários ou todos que ganhem entre 1000 e 2000 faça:

```
SELECT EMPNO, ENAME, JOB, SAL
  FROM EMP
 WHERE SAL BETWEEN 1000 AND 2000
    OR JOB = 'CLERK';
```

| EMPNO | ENAME | JOB | SAL |
|-------|--------|----------|----------|
| 7369 | SMITH | CLERK | 800.00 |
| 7499 | ALLEN | SALESMAN | 1,600.00 |
| 7521 | WARD | SALESMAN | 1,250.00 |
| 7654 | MARTIN | SALESMAN | 1,250.00 |
| 7844 | TURNER | SALESMAN | 1,500.00 |
| 7876 | ADAMS | CLERK | 1,100.00 |
| 7900 | JAMES | CLERK | 950.00 |
| 7934 | MILLER | CLERK | 1,300.00 |

Você pode combinar AND e OR na mesma expressão lógica. Quando AND e OR aparecer na mesma cláusula WHERE, todos os ANDs serão feitos primeiros e então todos os Ors serão feitos. Se AND não interfere sobre o OR a seguinte declaração SQL retornará todos os gerentes com salário acima de 1500, e todos os vendedores.

```
SELECT EMPNO, ENAME, JOB, SAL, DEPTNO
  FROM EMP
 WHERE SAL > 1500
    AND JOB = 'MANAGER'
    OR JOB = 'SALESMAN'
```

| EMPNO | ENAME | JOB | SAL | DEPTNO |
|-------|--------|----------|----------|--------|
| 7499 | ALLEN | SALESMAN | 1,600.00 | 30 |
| 7521 | WARD | SALESMAN | 1,250.00 | 30 |
| 7566 | JONES | MANAGER | 2,975.00 | 20 |
| 7654 | MARTIN | SALESMAN | 1,250.00 | 30 |
| 7698 | BLAKE | MANAGER | 2,850.00 | 30 |
| 7782 | CLARK | MANAGER | 2,450.00 | 10 |
| 7844 | TURNER | SALESMAN | 1,500.00 | 30 |

Se você quiser selecionar todos os gerentes e vendedores com salários acima de 1500 você deveria fazer:

```
SELECT EMPNO, ENAME, JOB, SAL, DEPTNO
  FROM EMP
 WHERE SAL > 1500
    AND (JOB = 'MANAGER' OR JOB = 'SALESMAN');
```

| EMPNO | ENAME | JOB | SAL | DEPTNO |
|-------|-------|----------|----------|--------|
| 7499 | ALLEN | SALESMAN | 1,600.00 | 30 |
| 7566 | JONES | MANAGER | 2,975.00 | 20 |
| 7698 | BLAKE | MANAGER | 2,850.00 | 30 |
| 7782 | CLARK | MANAGER | 2,450.00 | 10 |

O parêntese especifica a ordem na qual os operadores devem ser avaliados. No segundo exemplo, o operador OR é avaliado antes do AND.

Precedência dos Operadores

Todos operadores são organizados em uma hierarquia essa determina sua precedência. Numa Expressão, as operações são feitas na ordem de sua precedência, do maior para o menor. Onde os operadores de igual precedentes são usados próximos a outro, eles são feitos da esquerda para direita.

Todos os comparativos e Operadores SQL tem igual precedente:

1. =, !=, <, >, <=, >=, BETWEEN ... AND ..., IN, LIKE, IS NULL.
2. NOT (para inverter o resultado das expressões lógicas: WHERE NOT (SAL>2000))
3. AND
4. OR

Sempre que você estiver em dúvida sobre qual dos dois operadores será feito primeiro quando a expressão é avaliada, use parênteses para clarear seu significado e assegure o SQL*Plus o que você pretende.

Suponha que você queira encontrar todos os gerentes, em vários departamentos, e todos os escriturários no departamento 10 unicamente:

```
SELECT *  
  FROM EMP  
 WHERE JOB = 'MANAGER'OR (JOB = 'CLERK' AND DEPTNO = 10);
```

O parêntese acima não é necessário, por que o AND é precedente ao OR, mas ele esclarece o significado da expressão.

Resumo do SELECT.

Resumo as clausulas que cobrem o comando SELECT:

SELECT (DISTINCT) (*, COLUNA (SINÔNIMO), ...)

| | |
|------|--------|
| FROM | tabela |
|------|--------|

| | |
|-------|-----------------------|
| WHERE | condição ou condições |
|-------|-----------------------|

| | |
|----------|--------------------------------|
| ORDER BY | (coluna, expressão) (ASC/DESC) |
|----------|--------------------------------|

| | |
|--------|--------------------------------|
| SELECT | Seleciona no mínimo uma coluna |
|--------|--------------------------------|

| | |
|-----------------|-----------------------------------------------------------------------------------------------------------|
| Sinônimo(Alias) | Pode ser usado para colunas unicamente na lista do SELECT |
| * | Indica todas as colunas |
| DISTINCT | Pode ser usado para eliminar duplicações. |
| FROM tabela | Indica a tabela onde as colunas origina. |
| WHERE | Restringe a pesquisa para linhas que encontram a condição. Ele pode conter colunas, expressões e literais |
| AND/OR | Podem ser usados na clausula WHERE para construir mais complexa condições, AND tem prioridade sobre o OR. |
| () | Pode ser usado para forçar prioridade. |
| ORDER BY | Sempre o último. Especifica a ordem de ordenação. Uma ou mais colunas podem ser especificadas. |
| ASC | Ordem ascendente 'é padrão ordem de ordenação e não precisa ser especificado. |
| DESC | Inverte a ordem padrão de ordenação e deve ser especificada depois do nome da coluna. |

Exercício 1 - Introdução ao SQL, o comando SELECT

Esses exercícios são feitos para introduzir todos os tópicos vistos anteriormente nas leituras.

1- Selecione todas as informações da tabela SALGRADE.

| GRADE | LOSAL | HISAL |
|-------|-------|-------|
| ----- | ----- | ----- |
| 1 | 700 | 1200 |
| 2 | 1201 | 1400 |
| 3 | 1401 | 2000 |
| 4 | 2001 | 3000 |
| 5 | 3001 | 9999 |

2- Selecione todas as informações da tabela EMP.

| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|-------|-------|----------|-------|-----------|----------|--------|--------|
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| 7369 | SMITH | CLERK | 7902 | 13-JUN-83 | 800.00 | | 20 |
| 7499 | ALLEN | SALESMAN | 7698 | 15-AUG-83 | 1,600.00 | 300.00 | 30 |

| | | | | | | | |
|------|--------|-----------|------|-----------|----------|----------|----|
| 7521 | WARD | SALESMAN | 7698 | 26-MAR-84 | 1,250.00 | 500.00 | 30 |
| 7566 | JONES | MANAGER | 7839 | 31-OCT-83 | 2,975.00 | | 20 |
| 7654 | MARTIN | SALESMAN | 7698 | 05-DEC-83 | 1,250.00 | 1,400.00 | 30 |
| 7698 | BLAKE | MANAGER | 7839 | 11-JUN-84 | 2,850.00 | | 30 |
| 7782 | CLARK | MANAGER | 7839 | 14-MAY-84 | 2,450.00 | | 10 |
| 7788 | SCOTT | ANALYST | 7566 | 05-MAR-84 | 3,000.00 | | 20 |
| 7839 | KING | PRESIDENT | | 09-JUL-84 | 5,000.00 | | 10 |
| 7844 | TURNER | SALESMAN | 7698 | 04-JUN-84 | 1,500.00 | .00 | 30 |
| 7876 | ADAMS | CLERK | 7788 | 04-JUN-84 | 1,100.00 | | 20 |
| 7900 | JAMES | CLERK | 7698 | 23-JUL-84 | 950.00 | | 30 |
| 7902 | FORD | ANALYST | 7566 | 05-DEC-83 | 3,000.00 | | 20 |
| 7934 | MILLER | CLERK | 7782 | 21-NOV-83 | 1,300.00 | | 10 |

3-Liste todos os empregados que tenham salário entre 1000 e 2000.

| ENAME | DEPTNO | SAL |
|--------|--------|----------|
| ----- | ----- | ----- |
| - | | - |
| ALLEN | 30 | 1,600.00 |
| WARD | 30 | 1,250.00 |
| MARTIN | 30 | 1,250.00 |
| TURNER | 30 | 1,500.00 |
| ADAMS | 20 | 1,100.00 |
| MILLER | 10 | 1,300.00 |

4- Liste número e nome de departamentos em ordem de nome.

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

5- Mostrar todos os diferentes tipos de cargos.

| JOB |
|-----------|
| ----- |
| ANALYST |
| CLERK |
| MANAGER |
| PRESIDENT |
| SALESMAN |

6- Listar todos os detalhes dos empregados dos departamentos 10 e 20 em ordem de nome

| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|-------|--------|-----------|-------|-----------|----------|-------|--------|
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| 7876 | ADAMS | CLERK | 7788 | 04-JUN-84 | 1,100.00 | | 20 |
| 7782 | CLARK | MANAGER | 7839 | 14-MAY-84 | 2,450.00 | | 10 |
| 7902 | FORD | ANALYST | 7566 | 05-DEC-83 | 3,000.00 | | 20 |
| 7566 | JONES | MANAGER | 7839 | 31-OCT-83 | 2,975.00 | | 20 |
| 7839 | KING | PRESIDENT | | 09-JUL-84 | 5,000.00 | | 10 |
| 7934 | MILLER | CLERK | 7782 | 21-NOV-83 | 1,300.00 | | 10 |
| 7788 | SCOTT | ANALYST | 7566 | 05-MAR-84 | 3,000.00 | | 20 |
| 7369 | SMITH | CLERK | 7902 | 13-JUN-83 | 800.00 | | 20 |

7- Listar nome e cargo de todos os escriturários do departamento 20.

| ENAME | JOB |
|-------|-------|
| SMITH | CLERK |
| ADAMS | CLERK |

8- Mostre todos os nomes dos empregados os quais tenham TH ou LL.

| ENAME |
|--------|
| SMITH |
| ALLEN |
| MILLER |

9- Listar os seguintes detalhes para todos empregados que tenham um gerente.

| ENAME | JOB | SAL |
|--------|----------|----------|
| SMITH | CLERK | 800.00 |
| ALLEN | SALESMAN | 1,600.00 |
| WARD | SALESMAN | 1,250.00 |
| JONES | MANAGER | 2,975.00 |
| MARTIN | SALESMAN | 1,250.00 |
| BLAKE | MANAGER | 2,850.00 |
| CLARK | MANAGER | 2,450.00 |
| SCOTT | ANALYST | 3,000.00 |
| TURNER | SALESMAN | 1,500.00 |
| ADAMS | CLERK | 1,100.00 |
| JAMES | CLERK | 950.00 |
| FORD | ANALYST | 3,000.00 |
| MILLER | CLERK | 1,300.00 |

10- Mostrar nome e total da remuneração para todos os empregados

| ENAME | REMUNERATION |
|--------|--------------|
| SMITH | 9600 |
| ALLEN | 19500 |
| WARD | 15500 |
| JONES | 35700 |
| MARTIN | 16400 |
| BLAKE | 34200 |
| CLARK | 29400 |
| SCOTT | 36000 |
| KING | 60000 |
| TURNER | 18000 |
| ADAMS | 13200 |
| JAMES | 11400 |
| FORD | 36000 |
| MILLER | 15600 |

11- Mostrar todos os empregados que foram admitidos durante 1983

| ENAME | DEPTNO | HIREDATE |
|--------|--------|-----------|
| SMITH | 20 | 13-JUN-83 |
| ALLEN | 30 | 15-AUG-83 |
| JONES | 20 | 31-OCT-83 |
| MARTIN | 30 | 05-DEC-83 |
| ADAMS | 20 | 04-JUN-84 |
| FORD | 20 | 05-DEC-83 |
| MILLER | 10 | 21-NOV-83 |

12- Mostrar nome, salário anual e comissão de todos os vendedores que o salário mensal é maior que sua comissão. A saída deve ser ordenada pelo maior salário primeiro. Se duas ou mais empregados tiverem o mesmo salário ordenar por nome de empregado.

| ENAME | ANNUAL_SAL | COMM |
|--------|------------|--------|
| ALLEN | 19200 | 300.00 |
| TURNER | 18000 | .00 |
| WARD | 15000 | 500.00 |

13- Construa uma instrução que mostre desta forma os dados de todos os empregados:

Who, what and when

```
SMITH HAS HELD THE POSITION OF CLERK IN DEPT 20 SINCE 13-JUN-95
ALLEN HAS HELD THE POSITION OF SALESMAN IN DEPT 30 SINCE 15-AUG-83
```

14- Selecione todos os dados da tabela EMP, classificando-os por MGR. Se MGR for nulo coloque no fim da lista.

Funções

Nessa Unidade estamos introduzindo as Funções. Funções fazem a pesquisa de bloco mais potente, e são usadas para manipular valores. Nessa Unidade falaremos de funções numéricas e alfanuméricas. Funções de Data, funções de Conversão, e funções as quais operam sobre tipos de dados que discutimos na Unidade 5. Finalmente funções de grupo que falamos na Unidade 6.

Introdução a Funções

Funções são usadas para manipular dados. Elas aceitam um ou mais argumentos e retorna um valor. Um argumento é uma constante, refere-se a variável ou coluna. O formato para uma função é a seguinte:

função_nome (arumento1,argument2,...)

Funções podem ser usadas para:

- Cálculos sobre datas
- modificar valores de itens individuais
- manipular saída para grupos de linhas
- alterar formatos de datas para mostrá-los

Existem diferentes tipos de funções:

- ALFANUMÉRICAS
- NUMÉRICAS
- DATA
- CONVERSÃO
- FUNÇÕES QUE ACEITAM VÁRIOS TIPOS DE DADOS
- GRUPO

Algumas funções operam unicamente sobre linhas simples; outras manipulam grupo de linhas.

Funções de Linha Única

- age sobre cada linha retornada na pesquisa
- retorna um resultado por linha
- espera um ou mais argumento do usuário
- pode ser aninhada
- podem ser usadas com variáveis do usuário, colunas, expressões podem ser usadas por exemplo nas : cláusulas SELECT, WHERE, ORDER BY.

Explicação da notação

| <u>Notação</u> | <u>Significado</u> |
|-----------------------|---------------------------|
|-----------------------|---------------------------|

| | |
|----------|-----------------------------------------------------|
| Col | qualquer nome de coluna do Banco de Dados |
| Value | qualquer valor literal (alfanumérico/data/numérico) |
| n | representa um número |
| 'string' | representa a linha de caracter |
| chars | representa o número de caracteres especificados |
| date | representa uma coluna data ou valor de data |

Funções Alfanuméricas e Numéricas

Funções Alfanuméricas

Funções Alfanuméricas aceitam dados alfanuméricos e podem retornar alfanumérico ou valores numéricos. A função seguinte influencia na construção de valores alfanuméricos.

LOWER

LOWER(col/value) fornece valores alfanuméricos os quais estão em letra maiúscula ou minúscula e retornam em letra minúscula

Para mostrar o nome dos departamentos em letra minúscula e a constante SQL COURSE, faça:

```
SELECT LOWER(DNAME), LOWER('SQL COURSE')
FROM DEPT;
```

```
LOWER(DNAME) LOWER(SQL COURSE)
-----
research      sql course
sales         sql course
operations    sql course
accounting    sql course
```

UPPER

UPPER(col/value) fornece caracteres alfanuméricos, os quais estão em letra maiúscula ou minúscula e retornar em letra maiúscula.

No exemplo seguinte, a função UPPER força o usuário entrar em letra maiúscula.

```
SELECT ENAME
FROM EMP
WHERE ENAME = UPPER('&ENAME');
```

Enter value for ename : smith

```
ENAME
-----
SMITH
```

INITCAP

INITCAP(col/value) força a primeira letra da Palavras ser em maiúscula e o resto minúscula

```
INITCAP(DNAME) INITCAP(LOC)
-----
Accounting      New York
Research        Dallas
Sales           Chicago
Operations       Boston
```

```
SELECT INITCAP(DNAME), INITCAP(LOC)
FROM DEPT;
```

LPAD e RPAD

As funções LPAD e RPAD enchem valores alfanuméricos para tamanhos especificados.

LPAD(col/value,n,'character') preenche a coluna ou valor literal da esquerda para o total tamanho de n posições. Os principais espaços estão preenchidos com o 'character'. Se o character for omitido o valor padrão é espaços.

```
SELECT LPAD(DNAME,20,'*'), LPAD(DNAME,20), LPAD(DEPTNO,20,'. ')
FROM DEPT;
```

| LPAD(DNAME,20,'*') | LPAD(DNAME,20) | LPAD(DEPTNO,20,'. ') |
|--------------------|----------------|----------------------|
| *****RESEACH | RESEACH |20 |
| *****SALES | SALES |30 |
| *****OPERATIONS | OPERATIONS |40 |
| *****ACCOUTING | ACCOUNTING |10 |

RPAD(col/value,n,'character') preenche a coluna ou valor literal da direita para o total tamanho de n posições. Os espaços a direita são preenchidos com o 'character'. Se o 'character' for omitido o preenchimento fica em branco.

```
SELECT RPAD(DNAME,20,'*'), RPAD(DNAME,20), RPAD(DEPTNO,20,'. ')
FROM DEPT;
```

| RPAD(DNAME,20,'*') | RPAD(DNAME,20) | RPAD(DEPTNO,20,'. ') |
|--------------------|----------------|----------------------|
| RESEACH***** | RESEACH | 20..... |
| SALES***** | SALES | 30..... |
| OPERATIONS***** | OPERATIONS | 40..... |
| ACCOUTING***** | ACCOUNTING | 10..... |

Essa vez a segunda coluna é alinhada para a direita com brancos por padrão.

SUBSTR

A função seguinte assume os caracteres na linha estando numerados da esquerda para a direita começando com 1.

SUBSTR(col/value,pos,n) Retorna um linha de n caracteres da coluna ou valor literal, iniciando na posição número pos. Se n é omitido a linha é extraída da posição pos até o fim.

O próximo exemplo mostra o seguinte “sublinha”;

- quatro caracteres do literal ORACLE iniciando na segunda posição.
- conteúdo do Dname iniciando no segundo caracter.
- cinco caracteres do DNAME iniciando na terceira posição.

```
SELECT SUBSTR('ORACLE',2,4), SUBSTR(DNAME,2), SUBSTR(DNAME,3,5)
FROM DEPT;
```

| SUBSTR('ORACLE',2,4) | SUBSTR(DNAME,2) | SUBSTR(DNAME,3,5) |
|----------------------|-----------------|-------------------|
| ----- | ----- | ----- |

| | | |
|------|-----------|-------|
| RACL | ESEARCH | SEAC |
| RACL | ALES | LES |
| RACL | PERATIONS | ERATI |
| RACL | CCOUNTING | COUNT |

INSTR

INSTR(col/value,'character') encontra a primeira ocorrência do 'character'.

INSTR(col/value,'character',pos,n) encontra a posição do caracter do tamanho do 'character' na coluna ou valor literal iniciando na posição número pos

```
SELECT DNAME, INSTR(DNAME, 'A'), INSTR(DNAME, 'ES'), INSTR(DNAME, 'C', 1, 2)
FROM DEPT;
```

| DNAME | INSTR(DNAME, 'A') | INSTR(DNAME, 'ES') | INSTR(DNAME, 'C', 1, 2) |
|------------|-------------------|--------------------|-------------------------|
| ACCOUNTING | 1 | 0 | 3 |
| RESEACH | 5 | 2 | 0 |
| SALES | 2 | 4 | 0 |
| OPERATIONS | 5 | 0 | 0 |

LTRIM e RTRIM

LTRIM e RTRIM removem específicos caracteres de um linha.

LTRIM(col/value,'caractere(s)') removem da esquerda principalmente ocorrências de caracteres (ou combinação de caracteres específicos). Se o caracter não é especificado cortará os brancos da esquerda

```
SELECT DNAME, LTRIM(DNAME, 'A'), LTRIM, 'AS'), LTRIM(DNAME, 'ASOP')
FROM DEPT;
```

| DNAME | LTRIM(DNAME, 'A') | LTRIM(DNAME, 'AS') | LTRIM(DNAME, 'ASOP') |
|------------|-------------------|--------------------|----------------------|
| RESEARCH | RESEARCH | RESEARCH | RESEARCH |
| SALES | SALES | LES | LES |
| OPERATIONS | OPERATIONS | OPERATIONS | ERATIONS |
| ACCOUNTING | CCOUNTING | CCOUTING | CCOUTING |

RTRIM(col/value,'caractere(s)') remove da direita ocorrência de caracter(s) (ou combinações de caracteres específicos). Se os caracteres não forem especificados serão removidos os brancos.

```
SELECT DNAME, RTRIM(DNAME, 'G'), RTRIM, 'GHS'), RTRIM(DNAME, 'N')
FROM DEPT;
```

| DNAME | RTRM(DNAME, 'G') | RTRIM(DNAME, 'GHS') | RTRIM(DNAME, 'N') |
|------------|------------------|---------------------|-------------------|
| RESEARCH | RESEARCH | RESEARC | RESEARCH |
| SALES | SALES | SALE | SALES |
| OPERATIONS | OPERATIONS | OPERATION | OPERATIONS |
| ACCOUNTING | ACCOUNTING | ACCOUITIN | ACCOUITING |

RTRIM pode ser usada para ajudar na remoção de brancos ou caracter do final de um campo.

SOUNDEX

SOUNDEX(col/value) retorna uma linha de caracteres representando o som da palavra para um coluna ou um valor literal. Esta função retorna a fonética representação de uma palavra, você pode comparar palavras que tenham escrita diferente e sons iguais.

```
SELECT ENAME, SOUNDEX(ENAME)
FROM EMP
WHERE SOUNDEX(ENAME) = SOUNDEX('FRED');
```

| ENAME | SOUNDEX(ENAME) |
|-------|----------------|
| FORD | F630 |

LENGTH

LENGTH(col/value) retorna o número de caracteres na coluna ou valor literal.

```
SELECT LENGTH('SQL COURSE'), LENGTH(DEPTNO), LENGTH(DNAME)
FROM DEPT;
```

| LENGTH('SQL COURSE') | LENGTH(DEPTNO) | LENGTH(DNAME) |
|----------------------|----------------|---------------|
| 10 | 2 | 8 |
| 10 | 2 | 5 |
| 10 | 2 | 10 |
| 10 | 2 | 10 |

Note como a função INSTR, LENGTH retorna um valor numérico.

TRANSLATE e REPLACE

As funções TRANSLATE e REPLACE são usadas para substituir caracteres.

TRANSLATE(col/value,from,to) transforma caracteres de para. Mais de um caracter pode ser combinado. Todas as ocorrências *from* são substituídas com a correspondente caracter no *to*. Se o correspondente *to* não for digitado o *from* será removido

```
SELECT ENAME, TRANSLATE(ENAME, 'C', 'P'), JOB,
       TRANSLATE(JOB, 'AR', 'IT')
FROM EMP
WHERE DEPTNO = 10;
```

| ENAME | TRANSLATE(ENAME, 'C', 'P') | JOB |
|----------------------------|----------------------------|-----|
| TRANSLATE(JOB, 'AR', 'IT') | | |

```

-----
-
CLARK      PLARK      MENAGER    MINIGET
KING       KING       PRESIDENT  PTESIDENT
MILLER     MILLER     CLERK      CLETK

```

REPLACE

REPLACE(col/value,linha,linha_alterada)

Retorna o valor da coluna com toda a ocorrência da linha de alteração. Se a linha alterada for omitida toda a linha especificada será removida.

```

SELECT JOB, REPLACE (JOB, 'SALESMAN', 'SALESPERSON'),
       ENAME, REPLACE (ENAME, 'CO', 'PX')
FROM EMP;

```

| JOB | REPLACE (JOB, 'SALESMAN', 'SALESPERSON') | ENAME | REPLACE (ENAME, 'CO', 'PX') |
|----------|------------------------------------------|--------|-----------------------------|
| ANALYST | ANALYST | SCOTT | SPXTT |
| SALESMAN | SALESPERSON | TURNER | TURNER |
| SALESMAN | SALESPERSON | ALLEN | ALLEN |
| MANAGER | MANAGER | CLARK | CLARK |

A Função REPLACE é um complemento da função TRANSLATE que substitui caracteres um a um e o REPLACE substitui um linha por outra.

Aninhamento de Funções

Funções de linhas únicas podem ser aninhadas para várias finalidades. Se funções são aninhadas, elas são avaliadas de dentro para fora. Isto é por exemplo:

X(D(A(B(C(caracter)))))) ordem lógica de execução “C,B,A,D e X.”

Suponhamos que você queira encontrar o número de vezes que um determinado caracter ocorre em uma linha. Como você faria isso?

Você pode aninhar as funções LENGTH e TRANSLATE para realizar um requisitado resultado. O exemplo seguinte permite contar o número de Ss em uma linha:

```

SELECT DNAME, LENGTH (DNAME),
       LENGTH (DNAME) - LENGTH (TRANSLATE (DNAME, 'AS', 'A'))
FROM DEPT;

```

| DNAME | LENGTH (DNAME) | LENGTH (DNAME) - LENGTH (TRANSLATE (DNAME, 'AS', 'A')) |
|------------|----------------|--------------------------------------------------------|
| RESEARCH | 8 | 1 |
| SALES | 5 | 2 |
| OPERATIONS | 10 | 1 |
| ACCOUNTING | 10 | 0 |

Aqui estão os passos para realizar esse resultado:

1. Usa-se o LENGTH para identificar o número de caracteres da linha.
2. Então usa a função TRANSLATE para fazer todas as ocorrências de S sair da linha.

```
SELECT LENGTH (TRANSLATE (DNAME, 'AS', 'A'))
FROM DEPT;
```

```
LENGTH (TRANSLATE (DNAME, 'AS', 'A'))
```

```
-----
RESEARCH
ALE
OPERATION
ACCOUNTING
```

3. Note que A é modificado para A, e S não corresponde a nenhum caracter para ser alterado. Como S não tem nenhum caracter correspondente ele é removido da linha..
4. Agora subtrai o tamanho da linha retirado os Ss do tamanho da linha original.(com os Ss)

```
LENGTH (DNAME) -LENGTH (TRANSLATE (DNAME, 'AS', 'A'))
```

- 5- O Resultado é o número de ocorrências do caracter S na linha.

Funções Numéricas

As funções aceitam entrada de números e retornam valores numéricos.

ROUND

ROUND(col/value,n) arredonda um coluna, expressão ou valor para n casas decimais. Se n é omitido não tem casas decimais. Se n for negativo, os números para esquerda do decimal são arredondados.

```
SELECT      ROUND (45.923, 1),
            ROUND (45.923),
            ROUND (45.323, 1),
            ROUND (45.323, -1),
            ROUND (SAL/32, 2)
FROM EMP
WHERE DEPTNO = 10;
```

| ROUND (45.923, 1) | ROUND (45.923) | ROUND (45.323, 1) | ROUND (45.323, -1) | ROUND (SAL/32, 2) |
|-------------------|----------------|-------------------|--------------------|-------------------|
| 45.9 | 46 | 45.3 | 40 | 76.56 |
| 45.9 | 46 | 45.3 | 40 | 156.25 |
| 45.9 | 46 | 45.3 | 40 | 40.63 |

TRUNC

TRUNC(col/value.n) trunca a coluna, expressão ou valor para n casas decimais, ou se n é omitido não têm casas decimais. Se n é negativo os números para esquerda das casas decimais são truncados para zero.

```
SELECT      TRUNC (45.923, 1),
            TRUNC (45.923),
```

```

        TRUNC(45.323,1),
        TRUNC(45.323,-1),
        TRUNC(SAL/32,2)

```

```

FROM EMP
WHERE DEPTNO = 10;

```

| TRUNC(45.923,1) | TRUNC(45.923) | TRUNC(45.323,1) | TRUNC(45.323,-1) | TRUNC(SAL/32,2) |
|-----------------|---------------|-----------------|------------------|-----------------|
| 45.9 | 45 | 45.3 | 40 | 76.56 |
| 45.9 | 45 | 45.3 | 40 | 156.25 |
| 45.9 | 45 | 45.3 | 40 | 40.62 |

CEIL

CEIL(col/value) encontra o menor valor maior que ou igual para a coluna, expressão ou valor.

```

SELECT CEIL(SAL), CEIL(99.9), CEIL(101.76), CEIL(-11.1)
FROM EMP
WHERE SAL BETWEEN 3000 AND 5000;

```

| CEIL(SAL) | CEIL(99.9) | CEIL(101.76) | CEIL(-11.1) |
|-----------|------------|--------------|-------------|
| 3000 | 100 | 102 | -11 |
| 5000 | 100 | 102 | -11 |
| 3000 | 100 | 102 | -11 |

FLOOR

FLOOR(col/value) encontra o maior valor menor que ou igual para a coluna, expressão ou valor

```

SELECT FLOOR(SAL), FLOOR(99.9), FLOOR(101.76), FLOOR(-11.1)
FROM EMP
WHERE SAL BETWEEN 3000 AND 5000;

```

| FLOOR(SAL) | FLOOR(99.9) | FLOOR(101.76) | FLOOR(-11.1) |
|------------|-------------|---------------|--------------|
| 3000 | 99 | 101 | -12 |
| 5000 | 99 | 101 | -12 |
| 5000 | 99 | 101 | -12 |

POWER

POWER(col/value,n) eleva uma coluna, expressão ou valor por uma potência, n pode ser negativo mas deve ser um número, se não um erro será retornado

```

SELECT SAL, POWER(SAL,2), POWER(SAL,3), POWER(50,5)
FROM EMP
WHERE DEPTNO = 10;

```

| SAL | POWER(SAL,2) | POWER(SAL,3) | POWER(50,5) |
|---------|--------------|--------------|-------------|
| 2450.0 | 6002500 | 14706125000 | 312500000 |
| 5000.0 | 25000000 | 125000000000 | 312500000 |
| 1300.00 | 1690000 | 2197000000 | 312500000 |

SQRT

SQRT(col/value) encontra a raiz quadrada da coluna ou valor. Se a coluna ou valor for menor que zero será retornado nulo.

```
SELECT SAL, SQRT(SAL), SQRT(40), SQRT(COMM)
FROM EMP
WHERE COMM > 0;
```

| SAL | SQRT(SAL) | SQRT(40) | SQRT(COMM) |
|---------|------------|------------|------------|
| 1600.00 | 40 | 6.32455532 | 17.3205081 |
| 1250.00 | 35.3553391 | 6.32455532 | 22.3606798 |
| 1250.00 | 35.3553391 | 6.32455532 | 37.4165739 |

SIGN

SIGN(col/value) retorna -1 se a coluna, expressão ou valor é negativa ou zero e 1 se for positivo

```
SELECT SAL-COMM, SIGN(SAL-COMM), COMM-SAL, SIGN(COMM-SAL)
FROM EMP
WHERE DEPTNO = 30;
```

| SAL-COMM | SIGN(SAL-COMM) | COMM-SAL | SIGN(COMM-SAL) |
|----------|----------------|----------|----------------|
| 1300 | 1 | -1300 | -1 |
| 750 | 1 | -750 | -1 |
| -150 | -1 | 150 | 1 |
| 1500 | 1 | -1500 | -1 |

Freqüentemente a função SIGN é usada para testar um valor se é menor, maior ou igual a um segundo valor. O seguinte exemplo apresenta todos os empregados os quais o salário é maior que sua comissão.

```
SELECT ENAME, SAL, COMM
FROM EMP
WHERE SIGN(SAL-COMM) = 1;
```

| ENAME | SAL | COMM |
|--------|------|------|
| ALLEN | 1600 | 300 |
| WARD | 1250 | 500 |
| TURNER | 1500 | 0 |

ABS

ABS(col/value) encontra o valor absoluto de um coluna, expressão ou valor

```
SELECT SAL, COMM, COMM-SAL, ABS(COMM-SAL), ABS(-35)
FROM EMP
WHERE DEPTNO = 30;
```

| SAL | COMM | COMM-SAL | ABS(COMM-SAL) | ABS(-35) |
|-----|------|----------|---------------|----------|
|-----|------|----------|---------------|----------|

| | | | | |
|---------|---------|-------|------|----|
| 1600.00 | 300.00 | -1300 | 1300 | 35 |
| 1250.00 | 500.00 | -750 | 750 | 35 |
| 1250.00 | 1400.00 | 150 | 150 | 35 |
| 2850.00 | | | | 35 |
| 1500.00 | .00 | -1500 | 1500 | 35 |
| 950.00 | | | | 35 |

MOD

MOD(val1,val2) encontra o resto da divisão val1 por val2

```
SELECT SAL, COMM, MOD(SAL,COMM), MOD(100,40)
FROM EMP
WHERE DEPTNO = 30
ORDER BY COMM;
```

| SAL | COMM | MOD(SAL,COMM) | MOD(100,40) |
|----------|----------|---------------|-------------|
| 2,850.00 | | | 20 |
| 950.00 | | | 20 |
| 1,600.00 | 300.00 | 100 | 20 |
| 1,250.00 | 500.00 | 250 | 20 |
| 1,250.00 | 1,400.00 | 1250 | 20 |
| 1,500.00 | .00 | 1500 | 20 |

Exercício 2 - Usando Funções

Esses exercícios convêm o uso de funções não somente para o SELECT mas no WHERE e ORDER BY. Se colunas sinônimas são usadas no resultado, use então na clausula SELECT da sua declaração SQL.

1. Liste os nomes e salários incrementados de 15% dos empregados e também os valores mostrados inteiros.

| DEPTNO | ENAME | PCTSAL |
|--------|--------|--------|
| 20 | SMITH | 920 |
| 30 | ALLEN | 1840 |
| 30 | WARD | 1438 |
| 20 | JONES | 3421 |
| 30 | MARTIN | 1438 |
| 30 | BLAKE | 3278 |
| 10 | CLARK | 3818 |
| 20 | SCOTT | 3450 |
| 10 | KING | 5750 |
| 30 | TURNER | 1725 |
| 20 | ADAMS | 1265 |
| 30 | JAMES | 1093 |
| 20 | FORD | 3450 |
| 10 | MILLER | 1495 |

2. Fazer a seguinte saída.

```
EMPLOYEE_AND_JOB
```

```
-----
```

| | |
|--------|-----------|
| SMITH | CLERK |
| ALLEN | SALESMAN |
| WARD | SALESMAN |
| JONES | MANAGER |
| MARTIN | SALESMAN |
| BLAKE | MANAGER |
| CLARK | MANAGER |
| SCOTT | ANALYST |
| KING | PRESIDENT |
| TURNER | SALESMAN |
| ADAMS | CLERK |
| JAMES | CLERK |
| FORD | ANALYST |
| MILLER | CLERK |

3. Mostrar uma lista de todos os empregados com um identificador o qual é composto das primeiras duas letras de seu cargo, o meio de dois dígitos de seu número e o código soundex do seu nome.

| NAME | CODE |
|--------|----------|
| ----- | ----- |
| SMITH | CL36S530 |
| ALLEN | SA49A450 |
| WARD | SA52W630 |
| JONES | MA56J520 |
| MARTIN | SA65M635 |
| BLAKE | MA69B420 |
| CLARK | MA78C462 |
| SCOTT | AN78S300 |
| KING | PR83K520 |
| TURNER | SA84T656 |
| ADAMS | CL87A352 |
| JAMES | CL90J520 |
| FORD | AN90F630 |
| MILLER | CL93M460 |

4. Fazer uma ferramenta intensiva de pesquisa para listar os empregados com o cargo que o usuário entrar.

```
Enter value for job : clerk
```

| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|-------|--------|-------|-------|-----------|----------|-------|--------|
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| 7369 | SMITH | CLERK | 7902 | 13-JUN-83 | 800.00 | | 20 |
| 7876 | ADAMS | CLERK | 7788 | 04-JUN-84 | 1,100.00 | | 20 |
| 7900 | JAMES | CLERK | 7698 | 23-JUL-84 | 950.00 | | 30 |
| 7934 | MILLER | CLERK | 7782 | 21-NOV-83 | 1,300.00 | | 10 |

5. Imprima um lista dos nomes de departamentos centralizados. Assuma a coluna com a largura de 20 caracteres.

```
DEPARTMENT
-----
ACCOUNTING
OPERATIONS
RESEACH
SALES
```

6. Encontrar a primeira ocorrência de L no nome do empregado, e substituí-la por X.

| ENAME | FIRST_OCCURENCE_OF_ |
|-------|---------------------|
| ----- | ----- |
| | L |

| | |
|--------|--------|
| SMITH | SMITH |
| ALLEN | AXLEN |
| WARD | WARD |
| JONES | JONES |
| MARTIN | MARTIN |
| BLAKE | BXAKE |
| CLARK | CXARK |
| SCOTT | SCOTT |
| KING | KING |
| TURNER | TURNER |
| ADAMS | ADAMS |
| JAMES | JAMES |
| FORD | FORD |
| MILLER | MIXLER |

Funções de Data

Funções de data operam sobre datas do ORACLE. Todas as funções de datas retornam valores de tipo data exceto MONTHS_BETWEEN o qual retorna um valor numérico.

Armazenamento de Datas no ORACLE

As datas no ORACLE são armazenadas como números e com as seguintes informações:

- Século
- Ano
- Mês
- Dia
- Horas
- Minutos
- Segundos

O padrão de data mostrados nas pesquisas é DD-MON-YY.

Sysdate

Sysdate é uma coluna que retorna a data e horário corrente. Você pode usar o SYSDATE como uma outra coluna qualquer. Por exemplo, você pode mostrar data corrente selecionando o sysdate de uma tabela simulada chamada DUAL. A tabela DUAL é uma tabela do sistema e deve ser permitido acessá-la para todos os usuários. Ela contém uma coluna DUMMY e uma linha com o valor X. A tabela DUAL é usada quando você quer retornar apenas uma linha.

Para mostrar a data corrente:

```
SELECT SYSDATE FROM DUAL;
```

Você poderia facilmente selecionar o sysdate da tabela EMP, mas 14 sysdate seriam retornados.

Usando Operadores Aritméticos

Devido o fato das datas serem armazenadas como número, é possível fazer cálculos com datas usando operadores aritméticos tal como adição e subtração. Você pode adicionar e subtrair números constantes e bem como data de data.

As operações que você pode realizar são:

data + número Adicionando um número de dias em uma data, produzindo uma nova data

data - número subtraindo um número de dias de uma data, produzindo uma nova data

data - data subtraindo uma data de outra, produzindo um número de dias

data+número/24 adicionando um número de horas em uma data produzindo um nova data

```
SELECT HIREDATE, HIREDATE+7, HIREDATE-7, SYSDATE - HIREDATE
FROM   EMP
WHERE  HIREDATE LIKE '%JUN%';
```

| HIREDATE | HIREDATE+7 | HIREDATE-7 | SYSDATE-HIREDATE |
|-----------|------------|------------|------------------|
| 13-JUN-83 | 20-JUN-83 | 06-JUN-83 | 1982.70628 |
| 11-JUN-84 | 18-JUN-84 | 04-JUN-84 | 1618.70628 |
| 04-JUN-84 | 11-JUN-84 | 28-MAY-84 | 1625.70628 |
| 04-JUN-84 | 11-JUN-84 | 28-MAY-84 | 1625.70628 |

Subtraindo SYSDATE de HIREDATE coluna da tabela EMP retorna o números de dias que o empregado está admitido.

MONTHS_BETWEEN

MONTHS_BETWEEN(data1,data2)

encontra o número de meses entre data 1 e data2. O resultado pode ser positivo ou negativo. Se a data 1 for posterior a data2, então o resultado será positivo, se a data 1 for menor que a data 2 o resultado será negativo.

```
SELECT MONTHS_BETWEEN(SYSDATE, HIREDATE),
       MONTHS_BETWEEN('01-JAN-84', '05-NOV-88')
FROM   EMP
WHERE  MONTHS_BETWEEN(SYSDATE, HIREDATE) > 59;
```

| MONTHS_BETWEEN(SYSDATE, HIREDATE) | MONTHS_BETWEEN('01-JAN-84', '05-NOV-88') |
|-----------------------------------|------------------------------------------|
| 65.0873622 | -58.429332 |
| 63 | -58.429332 |
| 60.5067171 | -58.429332 |
| 59.3454267 | -58.429332 |
| 59.3454267 | -58.429332 |
| 59.8292977 | -58.429332 |

A parte não inteira do resultado representa um parcela do mês.

ADD_MONTHS

ADD_MONTHS(data,n) adiciona n números de meses na data. n deve ser inteiro e pode ser negativo.

```
SELECT HIREDATE, ADD_MONTHS(HIREDATE, 3), ADD_MONTHS(HIREDATE, -3)
FROM EMP
WHERE DEPTNO = 20;
```

| HIREDATE | ADD_MONTHS(HIREDATE, 3) | ADD_MONTHS(HIREDATE, -3) |
|-----------|-------------------------|--------------------------|
| 13-JUN-83 | 13-SEP-83 | 13-MAR-83 |
| 31-OCT-83 | 31-JAN-84 | 31-JUL-83 |
| 05-MAR-84 | 05-JUN-84 | 05-DEC-83 |
| 04-JUN-84 | 04-SEP-84 | 04-MAR-84 |
| 05-DEC-83 | 05-MAR-84 | 05-SEP-83 |

NEXT_DAY

NEXT_DAY(data1,'caracter') data do próximo dia especificado da semana(caracter) seguinte data.1. Caracter deve ser um número representado um dia, ou o dia da semana descrito em inglês.

```
SELECT HIREDATE, NEXT_DAY(HIREDATE, 'FRIDAY'),
       NEXT_DAY(HIREDATE, 6)
FROM EMP
WHERE DEPTNO = 20;
```

| HIREDATE | NEXT_DAY(HIREDATE, 'FRIDAY') | NEXT_DAY(HIREDATE, 6) |
|-----------|------------------------------|-----------------------|
| 14-MAY-84 | 18-MAY-84 | 18-MAY-84 |
| 09-JUL-84 | 13-JUL-84 | 13-JUL-84 |
| 21-NOV-83 | 25-NOV-83 | 25-NOV-83 |

LAST_DAY

LAST_DAY(data) encontra a data do ultimo dia do mês da data especificada

```
SELECT SYSDATE, LAST_DAY(SYSDATE), HIREDATE,
       LAST_DAY(HIREDATE), LAST_DAY('15-FEB-88')
FROM EMP
WHERE DEPTNO = 20;
```

| SYSDATE | LAST_DAY(SYSDATE) | HIREDATE | LAST_DAY(HIREDATE) | LAST_DAY('15-FEB-88') |
|-----------|-------------------|-----------|--------------------|-----------------------|
| 04-DEC-89 | 31-DEC-89 | 17-DEC-80 | 31-DEC-80 | 29-FEB-88 |
| 04-DEC-89 | 31-DEC-89 | 02-APR-81 | 30-APR-81 | 29-FEB-88 |
| 04-DEC-89 | 31-DEC-89 | 09-DEC-82 | 31-DEC-82 | 29-FEB-88 |
| 04-DEC-89 | 31-DEC-89 | 12-JAN-83 | 31-JAN-83 | 29-FEB-88 |
| 04-DEC-89 | 31-DEC-89 | 03-DEC-81 | 31-DEC-81 | 29-FEB-88 |

A função ROUND pode ser aplicada para datas.

ROUND(data) retorna a data com o horário em 12:00(meio-dia) Isso usamos quando comparamos datas que tenham diferentes horários.

ROUND(data,'MONTH') retorna o primeiro dia do mês da data, Se a data estiver na primeira metade do mês; se não retorna o primeiro do mês seguinte.

ROUND(data,'YEAR') retorna o primeiro dia do ano da data se data estiver na primeira metade do ano; se não retorna o primeiro do ano seguinte.

```
SELECT SYSDATE, ROUND(SYSDATE, 'MONTH'), ROUND(SYSDATE, 'YEAR')
FROM DUAL;
```

| SYSDATE | ROUND(SYSDATE, 'MONTH') | ROUND(SYSDATE, 'YEAR') |
|-----------|-------------------------|------------------------|
| 04-DEC-89 | 01-DEC-89 | 01-JAN-90 |

TRUNC

TRUNC(data,'character') encontra a data do primeiro dia do mês da data quando o caracter = 'MONTH'. Se o caracter = 'YEAR' ele encontra o primeiro dia do ano.

```
SELECT SYSDATE, TRUNC(SYSDATE, 'MONTH'), TRUNC(SYSDATE, 'YEAR')
FROM DUAL;
```

| SYSDATE | TRUNC(SYSDATE, 'MONTH') | TRUNC(SYSDATE, 'YEAR') |
|-----------|-------------------------|------------------------|
| 04-DEC-89 | 01-DEC-89 | 01-JAN-89 |

TRUNC é usado se você quiser remover o horário do dia. O horário contido no dia é removido por padrão.

Funções de Conversão

SQL possui um número de funções para controlar os tipos de conversão de dados. Essas funções de conversões converte um valor de um tipo de dado para outro.

TO_CHAR(número,data,'formato') converte números e datas para formatos alfanuméricos

TO_NUMER(caracter) converte alfanuméricos os quais possuem números para numéricos.

TO_DATE('caracter','formato') converte um alfanumérico representando uma data, para um valor de data de acordo com o formato especificado. Se o formato é omitido o formato padrão é 'DD=MON-YY'.

TO_CHAR

A função TO_CHAR é freqüentemente usada para modificar um formato de data padrão para um formato alternativo para mostrar.

TO_CHAR(data,'máscara') especifica que a data está sendo convertida para um novo formato na saída.

Para converter a data corrente do formato padrão (DD-MON-YY) para uma nova máscara.

```
SELECT TO_CHAR(SYSDATE, 'DAY, DDTH MONTH YYYY')
FROM DUAL;
```

```
TO_CHAR(SYSDATE, 'DAY, DDTH MONTH YYYY')
-----
TUESDAY      , 05TH SEPTEMBER 1989
```

Note que:

- A ‘máscara’ a qual deve estar entre aspas simples, pode ser incluída em vários formatos listados anteriormente. A coluna e ‘máscara’ deve ser separadas por uma vírgula.
- DAY e MONTH na saída são espaçados automaticamente com brancos no tamanho de 9 caracteres

Para remover um espaço em branco usar o FM(maneira de enchimento) prefixo:

```
SELECT TO_CHAR(SYSDATE, 'DAY, DDTH MONTH YYYY')
FROM DUAL;
```

```
TO_CHAR(SYSDATE, 'DAY, DDTH MONTH YYYY')
-----
TUESDAY,                05TH                SEPTEMBER                1989
```

- FM pode ser usado para suprimir zeros para o formato ddth, ex.: 05TH é alterado para 5TH
- A formato que a data será entrada é o formato que será mostrado.

TO_CHAR pode também ser usado para extrair o horário de um único dia, e mostrá-lo no especificado formato.

Para mostrar o formato de um dia:

```
SELECT TO_CHAR(SYSDATE, 'HH:MI:SS')
FROM DUAL;
```

```
TO_CHAR(SYSDATE, 'HH:MI:SS')
-----
08:16:24
```

A função TO_CHAR é também usada para converter um valor do tipo numérico para um valor do tipo alfanumérico.

TO_CHAR(numer, 'número máscara')

```
SELECT TO_CHAR(SAL, '$9,999')
FROM EMP;
```

```
TO_CHAR(SAL, '$9,999')
```

 \$1,000
 \$1,600
 \$1,250
 \$2,975

Note que os formatos das máscaras são opcionais. Se a ‘máscara’ é omitida, a data é convertida para um alfanumérico valor ORACLE que é padrão DD-MON-YY. Se o ‘máscara’ não é especificada, o número é convertido para alfanumérico. Então note que o formato modelo não afetam o valor interno que representam o valor das colunas. Eles somente afetam como a coluna é mostrada quando retirada com uma declaração SELECT.

Formatos de Data

| <u>Máscara</u> | <u>Significado</u> |
|-----------------------|-----------------------------------------------------------------------|
| SCC ou CC | Século, prefixo ‘S’ “BC” data com ‘-’ |
| YYYY ou SYYYY | Ano, prefixo ‘S’ “BC” data com ‘-’ |
| YYY ou YY ou Y | Último 3, 2 ou 1 dígito(s) do ano |
| Y,YYY | Ano com vírgula nessa posição |
| SYEAR ou YEAR | Ano, soletrado na saída ‘S’ prefixo “BC” data com ‘-’ |
| BC ou AD | BC/AD período |
| B.C. ou A.D. | BC/AD indicador com períodos |
| Q | Um quarto do Ano |
| MM | Mês |
| MONTH | nome do mês, espaçamento com brancos do tamanho de 9 caracteres |
| MON | nome do mês, 3 letras abreviadas |
| WW ou W | Semana do ano ou mês |
| DDD ou DD ou D | dia do ano, mês ou semana |
| DAY | nome do dia, espaçado com brancos com 9 caracteres de tamanho |
| DY | nome do dia, 3 letras abreviadas |
| J | data Juliana, o número de dias desde 31 dezembro 4713 antes de Cristo |
| AM ou PM | Indicador meridiano |
| A.M. ou P.M. | indicador meridiano com períodos |

| | |
|------------|-------------------------------------------------|
| HH ou HH12 | horas do dia (1-12) |
| HH24 | horas do dia (0-23) |
| MI | minuto |
| SS | segundos |
| SSSSS | segundos passado meia-noite(0-86399) |
| /.,etc. | pontuação é reproduzida no resultado |
| "..." | cotas de linhas são representadas no resultado. |

Os prefixos abaixo devem ser adicionados em frente aos códigos.

Fm (mode de enchimento) Prefixo para MONTH ou DAY suprime os espaçamentos em brancos, partindo um tamanho de uma variável, FM suprimirá zeros para o formato ddth. Não significa que em outros códigos uma segunda ocorrência for FM torne brancos os espaços de novo.

Os sufixos abaixo devem ser adicionados em frente dos códigos:

| | |
|--------------|----------------------------------------------------------|
| TH | número ordinal("DDTH" para "4TH") |
| SP | soletrando saída do número("DDSP" para "FOUR") |
| SPTH ou thsp | soletra o número ordinal na saída("DDSPTH"para "FOURTH") |

Nota: Os códigos são ferramentas sensíveis e afetaram a amostragem dos elementos da data:

| | |
|-------|--------|
| DAY | MONDAY |
| Day | Monday |
| Month | July |
| Ddth | 14th |
| DdTh | 14Th |

Formatos Numéricos

Os elementos do formato numérico modelo são:

| Máscara | Significado | Exemplo | |
|---------|---------------------------------------------------------------|-----------|---------|
| 9 | posição numérica (número de 9s determinam a largura mostrada) | 999999 | 1234 |
| 0 | mostra zeros | 0999999 | 001234 |
| \$ | mostra sinal de dólar | \$999999 | \$1234 |
| . | ponto decimal na posição especificada | 999999.99 | 1234.00 |
| , | vírgula na posição especificada | 999,999 | 1,234 |

| | | | |
|------|---------------------------------------------------------------|------------|-----------|
| MI | sinal de menos à direita(valores negativos) | 999999MI | 1234- |
| PR | parênteses para números negativos | 999999PR | <1234> |
| EEEE | notação científica(formato de conter quatro Es unicamente) | 99.999EEEE | 1.234E+03 |
| V | multiplica pela décima potência 10n(n = número 9s depois da V | 9999V99 | 123400 |
| B | mostra valores zero em branco, não zero | B9999.99 | 1234.00 |

TO_NUMBER

No seguinte exemplo a função TO_NUMBER é usada para transformar um número armazenado como um alfanumérico para um tipo numérico:

```
SELECT EMPNO, ENAME, JOB, SAL
FROM EMP
WHERE SAL > TO_NUMBER('1500');
```

| EMPNO | ENAME | JOB | SAL |
|-------|-------|-----------|-------|
| ---- | ----- | ----- | ----- |
| 7499 | ALLEN | SALESMAN | 1600 |
| 7566 | JONES | MANAGER | 2975 |
| 7698 | BLAKE | MANAGER | 2850 |
| 7782 | CLARK | MANAGER | 2450 |
| 7788 | SCOTT | ANALYST | 3000 |
| 7839 | KING | PRESIDENT | 5000 |
| 7902 | FORD | ANALYST | 3000 |

TO_DATE

Para mostrar todos os empregados admitidos em 4 de junho de 1984 (não formato padrão), nós podemos usar a função TO_DATE:

```
SELECT EMPNO, ENAME, HIREDATE
FROM EMP
WHERE HIREDATE = TO_DATE('June 4,1984','Month dd, yyyy');
```

| EMPNO | ENAME | HIREDATE |
|-------|--------|-----------|
| ---- | ----- | ----- |
| 7844 | TURNER | 04-JUN-84 |

O conteúdo é convertido para data e comparado com o valor de HIREDATE.

A função TO_DATE é frequentemente usada para suprir o valor ORACLE em um outro valor que o do padrão, Por exemplo, quando você insere um data, o ORACLE espera ser passado o valor no formato padrão DD-MON-YY. Se você não quer usar o formato padrão, você deve usar a função TO_DATE e apropriar o alternativo formato

Por exemplo:

Para entrar um linha na tabela EMP com a data não no formato padrão:

```
INSERT INTO EMP (EMPNO, DEPTNO, HIREDATE)
VALUES (7777, 20, TO_DATE('19/08/90', 'DD/MM/YY'));
```

O comando INSERT é comentado em detalhes mais adiante

Funções que Aceitam Vários Tipos de Entrada de Dados

DECODE

DECODE é a mais potente função do SQL. Ele facilita pesquisas condicionais fazendo o trabalho de ‘ferramentas’ ou comandos ‘se-então-se não’.

Sintaxe:

```
DECODE(col/expressão,
procurado1,resultado1,...,padrão)
```

Col/expressão é comparado com cada um dos valores procurado e retorna o resultado se a col/expressão é igual ao valor procurado. Se não for encontrada nenhum dos valores procurados, a função DECODE retorna o valor padrão. Se o valor padrão for omitido ele retornará um valor nulo.

Argumentos

DECODE deve ter no mínimo 4 parâmetros ou argumentos.

- COL/EXPRESSÃO - a nome da coluna ou expressão a ser avaliado.
- PROCURADO1 - o primeiro valor para ser testado
- RESULTADO1- o valor para ser retornado se o procurado1 for encontrado.
- PROCURADO1 e RESULTADO1 podem ser repetidos quantas vezes forem necessários.-(PROCURADO2,RESULTADO2, PROCURADO3,RESULTADO3,...)
- PADRÃO - o valor a ser retornado se nenhum procurado for encontrado.

Nota:

- col/expressão pode ser vários tipos de dados.
- PROCURADO deve ser um dado do tipo coluna ou expressão
- O valor retornado é forçado para alguns tipos de dados como o terceiro argumento(resultado1).

O seguinte exemplo decodifica os cargos dos tipos MANAGER e CLERK unicamente. Os outros cargos serão padrão alterados para UNDEFINED.

```
SELECT ENAME, JOB,
       DECODE(JOB, 'CLERK', 'WORKER',
               'MANAGER', 'BOSS',
               'UNDEFINED' DECODE_JOB
FROM EMP;
```

| ENAME | JOB | DECODE_JOB |
|--------|-----------|------------|
| ----- | ----- | ----- |
| SMITH | CLERK | WORKER |
| ALLEN | SALESMAN | UNDEFINED |
| WARD | SALESMAN | UNDEFINED |
| JONES | MANAGER | BOSS |
| MARTIN | SALESMAN | UNDEFINED |
| BLAKE | MANAGER | BOSS |
| CLARK | MANAGER | BOSS |
| SCOTT | ANALYST | UNDEFINED |
| KING | PRESIDENT | UNDEFINED |
| TURNER | SALESMAN | UNDEFINED |
| ADAMS | CLERK | WORKER |
| JAMES | CLERK | WORKER |
| FORD | ANALYST | UNDEFINED |
| MILLER | CLERK | WORKER |

Para mostrar a gratificação percentual dependendo do grau do salário:

```
SELECT GRADE,
DECODE (GRADE, '1', '15%',
           '2', '10%',
           '3', '8%',
           '5%') BONUS FROM SALGRADE;
```

| GRADE | BONUS |
|-------|-------|
| ----- | ----- |
| 1 | 15% |
| 2 | 10% |
| 3 | 8% |
| 4 | 5% |
| 5 | 5% |

Esse exemplo ilustra como a função decode, o valor retornado é forçado a ter um tipo de dado no terceiro argumento. Nós permitimos o usuário especificar a ordem na qual a informação empregado é mostrada por entrada de um valor na hora da execução

```
select * from emp
order by decode(&orderby,
                1,sal,
                2,ename
                sal);
Enter value for orderby: 2

ERROR at line 2: ORA-1722: invalid number
```

Note que esse comando causa um erro porque o tipo de dado de ename (alfanumérico) diferente que o do sal (numérico) o qual é o terceiro argumento. No exemplo abaixo, nós queremos retornar o salário incrementado de acordo com o tipo de cargo.

```
SELECT JOB, SAL, DECODE (JOB, 'ANALYST', SAL*1.1,
```

```

'CLERK', SAL*1.15,
'MANAGER', SAL*.095,
SAL)
FROM EMP;

```

NVL

NVL(col/valor,valor) converte um valor nulo para um valor desejado. Tipo de dados devem combinar(col/valor e valor).

```

SELECT SAL*12+NVL(COMM,0), NVL(COMM,1000), SAL*12+NVL(COMM,1000)
FROM EMP
WHERE DEPTNO = 10;

```

| SAL*12+NVL(COMM,0) | NVL(COMM,1000) | SAL*12+NVL(COMM,1000) |
|--------------------|----------------|-----------------------|
| 29400 | 1000 | 30400 |
| 60000 | 1000 | 61000 |
| 15600 | 1000 | 16600 |

GREATEST

GREATEST(col/valor1,col/valor2,...) retorna o maior da lista de valores. Todos os col/valores são convertidos para um valor antes da comparação.

```

SELECT GREATEST(1000,2000), GREATEST(SAL,COMM)
FROM EMP
WHERE DEPTNO = 30;

```

| GREATEST(1000,2000), | GREATEST(1000,2000), |
|----------------------|----------------------|
| 2000 | 1600 |
| 2000 | 1250 |
| 2000 | 1400 |
| 2000 | |
| 2000 | 1500 |

Nota: Na função GREATEST quando na lista de valores existe um valor nulo ele é considerado como o maior.

LEAST

LEAST(col/valor1,col/valor2,...) retorna o menor valor de um lista de valores. Todos os valores são convertidos antes da comparação.

```

SELECT LEAST(1000,2000), LEAST(SAL,COMM)
FROM EMP
WHERE DEPTNO = 30;

```

| LEAST(1000,2000), | LEAST(1000,2000), |
|-------------------|-------------------|
| 1000 | 300 |
| 1000 | 500 |
| 1000 | 1250 |
| 1000 | |

1000

0

Nota: Na função LEAST quando na lista de valores existe um valor nulo ele é considerado como o menor.

VSIZE

VSIZE(col/valor) retorna o número de bytes interno do ORACLE representando um col/valor.

```
SELECT DEPTNO, VSIZE(DEPTNO), VSIZE(HIREDATE), VSIZE(SAL), VSIZE(ENAME)
FROM EMP
WHERE DEPTNO = 10;
```

| DEPTNO | VSIZE (DEPTNO) | VSIZE (HIREDATE) | VSIZE (SAL) | VSIZE (ENAME) |
|--------|----------------|------------------|-------------|---------------|
| 10 | 2 | 7 | 3 | 5 |
| 10 | 2 | 7 | 2 | 4 |
| 10 | 2 | 7 | 2 | 6 |

Revisando Aninhamento de Funções

Relembrando que funções podem ser aninhadas em vários níveis, e que o interior do aninhamento é avaliado primeiro, trabalhamos com a última função externa. Ela é então seguida de um trilha de abertura e fechamento de parênteses, que deve ser o número de cada uma.

As funções abaixo tem sido aninhadas e são executadas como a seguir:

```
SELECT ENAME, NVL(TO_CHAR(MGR), 'UNMANAGEABLE')
FROM EMP
WHERE MGR IS NULL;
```

| ENAME | NVL (TO_CHAR (MGR) , 'UNMANAGEABLE') |
|-------|--------------------------------------|
| KING | UNMANAGEABLE |

1. MGR é um coluna convertida para alfanumérica com a função TO_CHAR.
2. A função NVL troca um MGR nulo por uma linha de caracteres 'UNMANAGEABLE'.

Funções aninhadas podem então serem usadas para mostrar a data de Quinta-feira que é de dois meses de hoje no formato de 'Day dd Month YYYY'.

```
SELECT SYSDATE,
TO_CHAR(NEXT_DAY(ADD_MONTHS(SYSDATE, 2), 'FRIDAY'), 'Day dd Month YYYY')
FROM DUAL;
```

| SYSDATE | TO_CHAR(NEXT_DAY(ADD_MONTHS(SYSDATE, 2), 'FRIDAY'), 'Day dd Month YYYY') |
|-----------|--------------------------------------------------------------------------|
| 04-DEC-89 | 09 February 1990 |

1. A função ADD_MONTHS adiciona dois meses para o corrente mês (dezembro).

2. A função NEXT_DAY encontra a Quinta-feira dois meses de SYSDATE.
3. A função TO_CHAR converte a coluna data para um tipo de alfanumérico na ordem para mostrar um não formato padrão de data 'Day dd Month YYYY'.

Exercício 3 – Mais Funções

Esse exercício convém de funções de linha única discutidas nessa Unidade, como também revisando algumas funções apresentadas anteriormente.

1. Mostrar nome e admissão dos empregados do departamento 20. Fazer com que a expressão fique com o nome de 'DATE_HIRED' com o tamanho de 80 colunas.

| ENAME | DATE_HIRED |
|-------|----------------------------|
| SMITH | June, Thirteenth 1983 |
| JONES | October, Thirty-First 1983 |
| SCOTT | March, Fifth 1984 |
| ADAMS | June, Fourth 1984 |
| FORD | December, Fifth 1983 |

2. Mostrar o nome com a admissão do empregado, e salário revisando a data. Assumindo a revisão da data um ano depois da admissão. Ordem de saída por revisão de data.

| ENAME | HIREDATE | REVIEW |
|--------|-----------|-----------|
| SMITH | 13-JUN-83 | 13-JUN-84 |
| ALLEN | 15-AUG-83 | 15-AUG-84 |
| JONES | 31-OCT-83 | 31-OCT-84 |
| MILLER | 21-NOV-83 | 21-NOV-84 |
| MARTIN | 05-DEC-83 | 05-DEC-84 |
| FORD | 05-DEC-83 | 05-DEC-84 |
| SCOTT | 05-MAR-84 | 05-MAR-85 |
| WARD | 26-MAR-84 | 26-MAR-85 |
| CLARK | 14-MAY-84 | 14-MAY-85 |
| TURNER | 04-JUN-84 | 04-JUN-85 |
| ADAMS | 04-JUN-84 | 04-JUN-84 |
| BLAKE | 11-JUN-84 | 11-JUN-85 |
| KING | 09-JUL-84 | 09-JUL-85 |
| JAMES | 23-JUL-84 | 23-JUL-85 |

3. Imprima uma lista de empregados mostrando justamente se o salário é maior que 1500. Se for exatamente igual 1500 mostre 'On Target', se menor mostre 'Below 1500'.

| ENAME | SALARY |
|--------|------------|
| ADAMS | Below 1500 |
| ALLEN | 1600 |
| BLAKE | 2850 |
| CLARK | 2450 |
| FORD | 3000 |
| JAMES | Below 1500 |
| JONES | 2975 |
| KING | 5000 |
| MARTIN | Below 1500 |
| MILLER | Below 1500 |
| SCOTT | 3000 |

| | |
|--------|------------|
| SMITH | Below 1500 |
| TURNER | On Target |
| WARD | Below 1500 |

4. Escreva uma pesquisa na qual retorne o dia da semana, para diferentes datas entradas no formato 'DD.MM.YY'.

Enter value for anydate: 12.11.88

DAY

SATURDAY

5. Escreva uma pesquisa que calcule o tempo que o empregado tem estado na companhia. Usar DEFINE para evitar as repetições típico das funções.

Enter value for employee_name: King

| ENAME | LENGTH_OF_SERVICE |
|-------|-------------------|
| ----- | ----- |
| KING | 4 YEARS 4 MONTHS |

6. Dado uma linha no formato 'nn/nn, muito que a primeiro e dois últimos caracteres sejam números, e que o meio é um alfanumérico '/'. Imprima a expressão 'YES' se válido, 'NO' se não válido. Use os seguintes valores para testar sua solução '12/34', '01/1^a.'99\88'.

VALUE VALID?

12/34 YES

7. Empregados admitidos antes da 15th dia do mês são espaçados na última Quinta do mês. Aqueles admitidos depois do 15th são espaçados da última quinta do mês seguinte. Imprima a lista dos empregados com suas datas de admissões e o primeiro dia de pagamento. Ordene por admissão.

| ENAME | HIREDATE | PAYDAY |
|--------|-----------|-----------|
| ----- | ----- | ----- |
| SMITH | 13-JUN-83 | 24-JUN-83 |
| ALLEN | 15-AUG-83 | 26-AUG-83 |
| JONES | 31-OCT-83 | 25-NOV-83 |
| MILLER | 21-NOV-83 | 30-DEC-83 |
| MARTIN | 05-DEC-83 | 30-DEC-83 |
| FORD | 05-DEC-83 | 30-DEC-83 |
| SCOTT | 05-MAR-84 | 30-MAR-84 |
| WARD | 26-MAR-84 | 27-APR-84 |
| CLARK | 14-MAY-84 | 25-MAY-84 |
| TURNER | 04-JUN-84 | 29-JUN-84 |
| ADAMS | 04-JUN-84 | 29-JUN-84 |
| BLAKE | 11-JUN-84 | 29-JUN-84 |
| KING | 09-JUL-84 | 27-JUN-84 |
| JAMES | 23-JUL-84 | 31-AUG-84 |

Funções de Grupo

Esta Unidade explica como resumir informações permitindo ser obtida por grupos de linhas e até o uso de grupos ou agregamento de funções. Nós discutiremos como você divide a linha em uma tabela em uma menor configuração, e como especificar critério de pesquisa para grupo de linhas.

GROUP BY

Funções de grupo operam sobre conjuntos de linhas. Elas retornam resultados baseados sobre um grupo de linhas, antes que um resultado por linha tenha retornado como uma função de linha única. Como padrão todas as linhas de um tabela são trilhadas como um grupo. A clausula GROUP BY da declaração do SELECT é usada para dividir as linhas em menores grupos.

As funções de grupos são listadas abaixo:

| <u>Função</u> | <u>Valor Retornado</u> |
|----------------------------|-------------------------------------------------------------------------------------------|
| AVG([DISTINCT/ALL]n) | Valor médio de n, ignorando os valores nulos. |
| COUNT([DISTINCT/ALL]expr*) | Contador * conta todas as linhas selecionadas, incluindo duplicadas e linhas nulas |
| MAX([DISTINCT/ALL]expr) | valor máximo da expressão |
| MIN([DISTINCT/ALL]expr) | valor mínimo da expressão |
| STDDEV([DISTINCT/ALL]n) | Desvio padrão de n, ignorando valores nulos. |
| SUM([DISTINCT/ALL]n) | Valor soma de n, ignorando valores nulos. |
| VARIANCE([DISTINCT/ALL],n) | variação de n, ignorando valores nulos. |

Todas as funções acima operam sobre um número de linhas (por exemplo, uma tabela inteira) e são portanto funções de GRUPO. DISTINCT faz uma função de grupo considerar valores não duplicados; ALL considera todos os valores sua declaração não é necessária. Os tipos de dados dos argumentos devem ser alfanuméricos, numéricos ou data onde a expressão é listada. Todas as funções de grupo exceto o COUNT(*) ignoram os valores nulos.

Usando Funções de Grupo:

AVG

Para calcular a média salarial dos empregados, faça:

```
SELECT AVG (SAL)
```

```
FROM EMP;
```

```
AVG (SAL)
-----
2073.21429
```

Note que as linhas da tabela EMP são trilhadas num único grupo.

MIN

Uma função de grupo pode ser usada para subconjunto de linhas de uma tabela usando a cláusula WHERE.

Para encontrar o mínimo salário ganho por um escriturário, faça:

```
SELECT MIN (SAL)
FROM EMP
WHERE JOB = 'CLEARK';
```

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

COUNT

Para Encontrar o número de empregados do departamento 20, faça:

```
SELECT COUNT (*)
FROM EMP
WHERE DEPTNO = 20;
```

```
COUNT (*)
-----
      5
```

Nota: A função COUNT usada dessa forma COUNT(1) tem o mesmo resultado que a acima e é mais rápida.

A cláusula GROUP BY

A cláusula GROUP BY pode ser usada para dividir as linhas de uma tabela em um menor grupo. Funções de grupo devem ser usadas para resumir informações por cada grupo.

Para calcular a média salarial de cada grupo de cargo, faça:

```
SELECT JOB, AVG (SAL)
FROM EMP
GROUP BY JOB;
```

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

| | |
|-----------|------------|
| CLERK | 1037.5 |
| MANAGER | 2758.33333 |
| PRESUDENT | 5000 |
| SALESMAN | 1400 |

Excluindo linhas quando estiver Usando o GROUP BY

Linhas devem ser excluídas com a clausula WHERE, antes da divisão por grupos. Para mostrar a média salarial para cada cargo excluindo os gerentes, faça:

```
SELECT JOB, AVG(SAL)
FROM EMP
WHERE JON <> 'MANAGER'
GROUP BY JOB;
```

| JOB | AVG (SAL) |
|-----------|-----------|
| ----- | ----- |
| ANALYST | 3000 |
| CLERK | 1037.5 |
| PRESUDENT | 5000 |
| SALESMAN | 1400 |

Grupos dentro de Grupos

Nós podemos então usar a clausula GROUP BY para prover resultados para grupos dentro de grupos.

Para mostrar a media salarial mensal faturado por cada cargo dentro de um departamento, faça:

```
SELECT DEPTNO, JOB, AVG(SAL) FROM EMP
GROUP BY DEPTNO, JOB;
```

| DEPTNO | JOB | AVG (SAL) |
|--------|-----------|-----------|
| ----- | ----- | ----- |
| 10 | CLERK | 1300 |
| 10 | MANAGER | 2450 |
| 10 | PRESIDENT | 5000 |
| 20 | ANALYST | 3000 |
| 20 | CLERK | 950 |
| 20 | MANAGER | 2975 |
| 30 | CLERK | 950 |
| 30 | MANAGER | 2850 |
| 30 | SALESMAN | 1400 |

Funções de Grupo e Resultados Individuais

A seguinte declaração SQL retorna o máximo salário para cada grupo. O resultado não é significativo porque o cargo não é mostrado no resultado.

```
SELECT MAX(SAL)
```

```
FROM EMP
GROUP BY JOB;
```

```
      MAX (SAL)
-----
3000
1300
2975
5000
1600
```

A mostra do cargo é opcional, mas a pesquisa fica um pouco sem sentido sem ele.

```
      MAX (SAL)  JOB
-----
3000            ANALYST
1300            CLERK
2975            MANAGER
5000            PRESIDENT
1600            SALESMAN
```

Suponha na mente a seguinte regra quando usar funções de grupo:

Se você inclui uma função de grupo no comando SELECT, você não deve selecionar resultados que não estejam declarados no GROUP BY.

Por exemplo:

```
SELECT DEPTNO, MIN(SAL)
FROM EMP;
```

```
ERROR at line 1: ORA-0937: not single row set function.
```

O comando é inválido porque DEPTNO tem um valor para cada linha da tabela, enquanto MIN(SAL) tem um valor para tabela inteira.

Para corrigir o erro, nós devemos agrupar o item individual:

```
SELECT DEPTNO, MIN(SAL)
FROM EMP;
```

```
DEPTNO  MIN (SAL)
-----
10      1300
20      800
30      950
```

DEPTNO no exemplo acima, não permanece um valor individual ele é um nome de um grupo. Portanto existe uma regra para usar a função GROUP BY. Se existir mais de uma coluna na declaração SELECT, ela(s) devem ser transformadas em nome de Grupo colocando-as na clausula WHERE.

A cláusula HAVING

Use a cláusula HAVING se você quiser especificar o qual grupo será mostrado.

Para mostrar a média salarial para todos os departamentos que tiverem mais de três empregados, faça:

```
SELECT    DEPTNO, AVG(SAL)
FROM      EMP
GROUP BY  DEPTNO
HAVING    COUNT(1) > 3;
```

| DEPTNO | AVG(SAL) |
|--------|-----------|
| 20 | 2175 |
| 30 | 1566.6667 |

Para mostrar só os cargos, onde o máximo salário é maior ou igual a \$3000, faça:

```
SELECT JOB, MAX(SAL)
FROM      EMP
HAVING    MAX(SAL) >= 3000
GROUP BY  JOB;
```

| JOB | MAX(SAL) |
|-----------|----------|
| ANALYST | 3000 |
| PRESIDENT | 5000 |

Nota:

A cláusula HAVING deve preceder uma cláusula GROUP BY, é recomendado que seja colocado primeiramente pois é mais lógico. Grupos são formados e funções de grupos são calculadas antes da cláusula HAVING é aplicado para selecionar a saída dos grupos.

A cláusula WHERE não pode ser usada para restringir itens de grupo. A seguinte declaração da cláusula WHERE é errada.

```
SELECT DEPTNO, AVG(SAL)
FROM EMP
WHERE AVG(SAL) > 2000
GROUP BY DEPTNO;
```

ERROR at line 3: ORA-0934: set function is not allowed here

Você pode unicamente usar WHERE para restringir linhas individuais. Para restringir colunas de grupos usa-se a cláusula HAVING.

```
SELECT DEPTNO, AVG(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING AVG(SAL) > 2000;
```

| DEPTNO | AVG (SAL) |
|--------|------------|
| 10 | 2916.66667 |
| 20 | 2175 |

Nota:

Você pode excluir todos os gerentes usando a cláusula WHERE quando estiver agrupando por cargo.

```
SELECT JOB, AVG(SAL)
FROM EMP
WHERE JOB <> 'MANAGER'
GROUP BY JOB;
```

| JOB | AVG (SAL) |
|-----------|-----------|
| ANALYST | 3000 |
| CLERK | 1037.5 |
| PRESIDENT | 5000 |
| SALESMAN | 1400 |

A Ordem das cláusulas na declaração SELECT.

| | |
|-----------------|------------------------------------|
| SELECT | <i>coluna(s)</i> |
| FROM | <i>tabela(s)</i> |
| WHERE | <i>condição linha</i> |
| GROUP BY | <i>coluna(s)</i> |
| HAVING | <i>condição de grupo de linhas</i> |
| ORDER BY | <i>coluna(s);</i> |

SQL avaliações:

- | | |
|-------------|-----------------------------------------------------------------------------|
| 1. WHERE | - para estabelecer linhas candidatas (não pode conter funções de grupos. |
| 2. GROUP BY | - para configurar grupos |
| 3. HAVING | para selecionar grupos para mostrar |

Exercício 4 - Funções de Grupo

No final desses exercícios você já se familiarizou com o uso das funções de Grupo e estará selecionando grupo de dados. Note as colunas sinônimas usadas.

1. Encontrar o mínimo salário dos empregados.

```
MINIMUM
```

```
-----
      800
```

2. Encontrar o mínimo, máximo e a média dos salários de todos os empregados.

```
MAX (SAL)    MIN (SAL)    AVG (SAL)
-----
5000         800         2073.21429
```

3. Listar o mínimo e máximo salário para os tipos de cargo.

```
JOB          MAXIMUM      MINIMUM
-----
ANALYST      3000             3000
CLERK        1300             1300
MANAGER      2975             2975
PRESIDENT    5000             5000
SALESMAN     1600             1600
```

4. Encontrar saída quantos gerentes sem listagem.

```
MANAGERS
-----
      3
```

5. Encontrar a média salarial e média total remuneração para cada cargo.
Lembre-se vendedores recebem comissão.

```
JOB          MAXIMUM      MINIMUM
-----
ANALYST      3000             36000
CLERK        1037.5           12450
MANAGER      2758.33333       33100
PRESIDENT    5000             60000
SALESMAN     1400             17350
```

6. Encontrar saída a diferença entre maior e menor salário.

```
DIFFERENCE
-----
      4200
```

7. Encontrar todos os departamentos os quais tem mais que 3 empregados

```
DEPTNO      COUNT (1)
-----
20          5
30          6
```

8. Cheque se todos os números de empregados são únicos.

9. Liste o menor espaçamento empregados trabalham para cada gerente. Exclua os grupos onde o mínimo salário é menor que 1000. Ordene pelo salário.

| MGR | MIN (SAL) |
|------|-----------|
| 7788 | 1100 |
| 7782 | 1300 |
| 7839 | 2450 |
| 7566 | 3000 |
| | 5000 |

Extraindo Dados de Mais de uma Tabela

Ligações (Joins)

Uma ligação é usada quando a pesquisa SQL requer dados de mais de uma tabela do Banco de Dados. Linhas em uma tabela devem ser ligadas a outras linhas de acordo com o valor comum existente na coluna correspondente. Existem dois tipos principais de condições de ligações:

- Equi-join
- Non-equi-join

Equi-Join

Na ordem para descobrir, manualmente, qual departamento vários empregados estão, nós comparamos a coluna DEPTNO da tabela EMP com a mesmo valor de DEPTNO na tabela DEPT. O relacionamento entre a tabela EMP e a DEPT é um equi-join, em que o valor da coluna DEPTNO seja igual para ambas as tabelas (o = operador de comparação é usado).

Uma condição de ligação é especificada na cláusula WHERE:

```
SELECT      coluna(s)
FROM        tabela(s)
WHERE       condição de ligação
```

Para ligar as duas tabelas EMP e DEPT, faça:

```
SELECT      ENAME, JOB, DNAME
FROM        EMP, DEPT
WHERE       EMP.DEPTNO = DEPT.DEPTNO;
```

| ENAME | JOB | DNAME |
|--------|-----------|------------|
| CLARK | MANAGER | ACCOUNTING |
| MILLER | CLERK | ACCOUNTING |
| KING | PRESIDENT | ACCOUNTING |
| SMITH | CLERK | RESEARCH |
| SCOTT | ANALYST | RESEARCH |

| | | |
|--------|----------|----------|
| JONES | MANAGER | RESEARCH |
| ADAMS | CLERK | RESEARCH |
| FORD | ANALYST | RESEARCH |
| ALLEN | SALESMAN | SALES |
| BLAKE | MANAGER | SALES |
| TURNER | SALESMAN | SALES |
| JAMES | CLERK | SALES |
| MARTIN | SALESMAN | SALES |
| WARD | SALESMAN | SALES |

Você percebe que todos os empregados tem seu respectivo nome de departamento. As linhas da EMP são combinadas com a da DEPT e só retornaram se o valor do MP.DEPTNO e DEPT.DEPTNO forem iguais. Note que a condição de ligação especifica o nome da tabela antes do nome da coluna. Isso é requerido quando o nome da coluna é o mesmo em ambas as tabelas. Ele é necessário para o ORACLE saber qual a coluna que ele está se referindo. Esses requerimento é então aplicado para colunas as quais são ambíguas nas clausulas SELECT ou ORDER BY.

Para distinguir entre a coluna DEPTNO na EMP e uma na DEPT, faça:

```
SELECT      DEPT.DEPTNO, ENAME, JOB, DNAME
FROM        EMP, DEPT
WHERE       EMP.DEPTNO = DEPT.DEPTNO
ORDER BY    DEPT.DEPTNO;
```

| DEPTNO | ENAME | JOB | DNAME |
|--------|--------|-----------|------------|
| ----- | ----- | ----- | ----- |
| 10 | CLARK | MANAGER | ACCOUNTING |
| 10 | MILLER | CLERK | ACCOUNTING |
| 10 | KING | PRESIDENT | ACCOUNTING |
| 20 | SMITH | CLERK | RESEARCH |
| 20 | SCOTT | ANALYST | RESEARCH |
| 20 | JONES | MANAGER | RESEARCH |
| 20 | ADAMS | CLERK | RESEARCH |
| 20 | FORD | ANALYST | RESEARCH |
| 30 | ALLEN | SALESMAN | SALES |
| 30 | BLAKE | MANAGER | SALES |
| 30 | TURNER | SALESMAN | SALES |
| 30 | JAMES | CLERK | SALES |
| 30 | MARTIN | SALESMAN | SALES |
| 30 | WARD | SALESMAN | SALES |

Note que cada número de departamento da tabela DEPT está ligado com o número de departamento da EMP. Por exemplo, três empregados trabalham no departamento 10 - ACCOUNTING - só existem três ocorrências.

Usando Tabelas com Sinônimos

Pode ser tedioso repetir o nome inteiro de um tabela em uma coluna. Nomes temporários (ou sinônimos) podem ser usados na clausula FROM. Estes nomes temporários valem unicamente para a declaração de SELECT corrente. Tabelas sinônimas devem então ser declaradas na clausula SELECT. Isso agiliza a pesquisa em que a declaração contém muitas informações.

Tabelas Sinônimas estão sendo usadas na seguinte declaração:

```

SELECT      E.ENAME, D.DEPTNO, D.DNAME
FROM        EMP E, DEPT D
WHERE       E.DEPTNO = D.DEPTNO
ORDER BY    D.DEPTNO;
```

Tabelas Sinônimas podem ter 30 caracteres de largura, mas os nomes curtos são melhores. Se um sinônimo for usado para uma determinada tabela na cláusula FROM então todas as declarações do SELECT deve usar esse sinônimo.

Nota:

Quando a condição de ligação é inválida ou omitida completamente o resultado é um PRODUTO, e todas combinações de linhas serão mostradas. Um produto cuida para gerar um grande número de linhas, e o resultado é raramente usado. Você sempre deve incluir a condição de ligação no WHERE a menos que você não tenha nenhuma ligação entre as tabelas envolvidas

Nom-Equi-Join

O relacionamento entre as tabelas EMP e SALGRADE é um nom-equi-join, em que a coluna na EMP corresponde direto a uma coluna na SALGRADE. O relacionamento é obtido usando um outro operador que o igual(=). Para avaliar um grau do salário de um empregado é necessário que o salário esteja entre o menor e maior faixa de salário.

O operador BETWEEN é usado para construir a condição, faça:

```

SELECT      E.ENAME, E.SAL, S.GRADE
FROM        EMP E, SALGRADE S
WHERE       E.SAL BETWEEN S.LOSAL AND S.HISAL;
```

| ENAME | SAL | GRADE |
|--------|----------|-------|
| ----- | ----- | ----- |
| SMITH | 800.00 | 1 |
| ADAMS | 1,100.00 | 1 |
| JAMES | 950.00 | 1 |
| WARD | 1,250.00 | 2 |
| MARTIN | 1,250.00 | 2 |
| MILLER | 1,300.00 | 2 |
| ALLEN | 1,600.00 | 3 |
| TURNER | 1,500.00 | 3 |
| JONES | 2,975.00 | 4 |
| BLAKE | 2,850.00 | 4 |
| CLARK | 2,450.00 | 4 |
| SCOTT | 3,000.00 | 4 |
| FORD | 3,000.00 | 4 |
| KING | 5,000.00 | 5 |

Outros operadores como <= e >= podem ser usados. Lembre-se de especificar o menor valor primeiro, e o maior no final quando usar BETWEEN. Outra vez as tabelas sinônimas foram usadas, não porque pode haver colunas ambíguas, mas por causa da performance.

Regras para Ligações de Tabelas.

Na ordem para três tabelas é necessário no mínimo duas condições de ligações.. Para quatro é necessário no mínimo três condições de ligações.

Uma regra simples:

O número de tabelas menos um é igual ao número de condições de ligações. Essa regra não se aplica se sua tabela tiver uma Primary Key que contém mais de uma coluna.

Resumo da Sintaxe

```
SELECT [DISTINCT] coluna(s), expr, alias...
FROM tabelas [alias]...
WHERE [condição de ligação]...
AND   [condição de linha]...
OR    [outras condições de linhas]..
GROUP BY [expr/coluna]
HAVING [grupo de condições]
ORDER BY [expr/coluna] [DESC/ASC]
```

Nota:

- você pode especificar condições de ligações e outras condições juntas

Exercício 5 - Simples Ligações (Join)

Esses exercícios são feitos para praticar a experiência em extrair dados de mais de uma tabela, e também inclui tópicos que foram vistos anteriormente.

1. Mostra todos os nomes dos empregados e o nome de seus departamentos em ordem de nome de departamento.

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

2. Mostrar o nome de todos os empregados, nome e número do departamento:

| ENAME | DEPTNO | DNAME |
|--------|--------|------------|
| ----- | ----- | ----- |
| CLARK | 10 | ACCOUNTING |
| MILLER | 10 | ACCOUNTING |
| KING | 10 | ACCOUNTING |
| SMITH | 20 | RESEARCH |
| SCOTT | 20 | RESEARCH |
| JONES | 20 | RESEARCH |
| ADAMS | 20 | RESEARCH |
| FORD | 20 | RESEARCH |
| ALLEN | 30 | SALES |
| BLAKE | 30 | SALES |
| TURNER | 30 | SALES |
| JAMES | 30 | SALES |
| MARTIN | 30 | SALES |
| WARD | 30 | SALES |

3. Mostrar o nome, localização e departamento dos empregados que tem o salário maior que 1500 por mês.

| ENAME | LOCATION | DNAME |
|-------|----------|------------|
| ----- | ----- | ----- |
| CLARK | NEW YORK | ACCOUNTING |
| KING | NEW YORK | ACCOUNTING |
| JONES | DALLAS | RESEARCH |
| FORD | DALLAS | RESEARCH |
| SCOTT | DALLAS | RESEARCH |
| ALLEN | CHICAGO | SALES |
| BLAKE | CHICAGO | SALES |

4. Proceder uma lista mostra o salário e grau do salário do empregado:

| ENAME | JOB | SAL | GRADE |
|--------|-----------|----------|-------|
| ----- | ----- | ----- | ----- |
| SMITH | CLERK | 800.00 | 1 |
| ADAMS | CLERK | 1,100.00 | 1 |
| JAMES | CLERK | 950.00 | 1 |
| WARD | SALESMAN | 1,250.00 | 2 |
| MARTIN | SALESMAN | 1,250.00 | 2 |
| MILLER | CLERK | 1,300.00 | 2 |
| ALLEN | SALESMAN | 1,600.00 | 3 |
| TURNER | SALESMAN | 1,500.00 | 3 |
| JONES | MANAGER | 2,975.00 | 4 |
| BLAKE | MANAGER | 2,850.00 | 4 |
| CLARK | MANAGER | 2,450.00 | 4 |
| SCOTT | ANALYST | 3,000.00 | 4 |
| FORD | ANALYST | 3,000.00 | 4 |
| KING | PRESIDENT | 5,000.00 | 5 |

5. Mostrar somente os empregados de grau 3:

| ENAME | JOB | SAL | GRADE |
|--------|----------|----------|-------|
| ----- | ----- | ----- | ----- |
| ALLEN | SALESMAN | 1,600.00 | 3 |
| TURNER | SALESMAN | 1,500.00 | 3 |

6. Mostrar todos os empregados de Dallas

| ENAME | SAL | LOCATION |
|-------|----------|----------|
| ----- | ----- | ----- |
| SMITH | 800.00 | DALLAS |
| JONES | 2,975.00 | DALLAS |
| BLAKE | 2,850.00 | DALLAS |
| SCOTT | 3,000.00 | DALLAS |
| ADAMS | 1,100.00 | DALLAS |

7. Listar nome, cargo, salário, grau e nome do departamento para todos os empregados da companhia exceto os escriturários. Ordene pelo salário, mostrando o maior primeiro.

| ENAME | JOB | SAL | GRADE | DNAME |
|--------|-----------|----------|-------|------------|
| KING | PRESIDENT | 5,000.00 | 5 | ACCOUNTING |
| FORD | ANALYST | 3,000.00 | 4 | RESEARCH |
| SCOTT | ANALYST | 3,000.00 | 4 | RESEARCH |
| JONES | MANAGER | 2,975.00 | 4 | RESEARCH |
| BLAKE | MANAGER | 2,850.00 | 4 | SALES |
| CLARK | MANAGER | 2,450.00 | 4 | ACCOUNTING |
| ALLEN | SALESMAN | 1,600.00 | 3 | SALES |
| TURNER | SALESMAN | 1,500.00 | 3 | SALES |
| MARTIN | SALESMAN | 1,250.00 | 2 | SALES |
| WARD | SALESMAN | 1,250.00 | 2 | SALES |

8. Listar o seguinte detalhe por empregado quem ganha \$36000 em um ano ou quem são escriturários.

| ENAME | JOB | ANNUAL_SAL | DEPTNO | DNAME | GRADE |
|--------|---------|------------|--------|------------|-------|
| FORD | ANALYST | 36000 | 20 | RESEARCH | 4 |
| SCOTT | ANALYST | 36000 | 20 | RESEARCH | 4 |
| MILLER | CLERK | 15600 | 10 | ACCOUNTING | 2 |
| JAMES | CLERK | 11400 | 30 | SALES | 1 |
| ADAMS | CLERK | 13200 | 20 | RESEARCH | 1 |
| SMITH | CLERK | 9600 | 20 | RESEARCH | 1 |

Outros Métodos de Ligação

Ligações Externas (Outer Join)

Se uma linha não satisfizer a condição de ligação, a linha não aparecerá no resultado da pesquisa. De fato no equi-join da EMP e DEPT, o departamento 40 não aparece. Isso porque não existe empregados no departamento 40.

Ligação Externa

| EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|-------|-------|-----|-----|----------|-----|------|--------|
| | | | | | | | |

| DEPTNO | DNAME | LOC |
|--------|-------|-----|
| | | |

40

OPERATIONS

BOSTON

..Não existe nenhum empregado no departamento 40, mas ele pode ser ligado com uma linha nula...

O que falta das linhas pode ser retornado se uma ligação externa (outer join) é usada na condição de ligação. O operador é um sinal de mais entre parênteses (+), e é colocado do lado da ligação a qual é deficiente a informação. O operador tem o efeito de criar uma ou mais linha nula, para que uma ou mais linha que não tenha ligação na tabela possa ser ligada. Uma ou mais linha nula será criada para todas as ligações que não possui resultado.

```
SELECT E.ENAME, D.DEPTNO, D.DNAME
FROM   EMP E, DEPT D
WHERE  E.DEPTNO (+) = D.DEPTNO
AND    D.DEPTNO IN (30,40)
```

| ENAME | DEPTNO | DNAME |
|--------|--------|------------|
| ----- | ----- | ----- |
| ALLEN | 30 | SALES |
| BLAKE | 30 | SALES |
| TURNER | 30 | SALES |
| JAMES | 30 | SALES |
| MARTIN | 30 | SALES |
| WARD | 30 | SALES |
| | 40 | OPERATIONS |

O operador de ligação externa pode unicamente aparecer sobre um dos lados da expressão, do lado que tem informação faltando. Ele retorna aquelas linhas de uma tabela que não diretamente se iguala com a outra tabela.

Ligando uma Tabela com Ela mesma

É possível usando nomes de tabelas (sinônimas) para ligar uma tabela com ela mesma, com se ela fosse duas tabelas separadas. Isso permite linhas de uma tabela ligar com linhas na mesma tabela.

A seguinte pesquisa mostra todos os empregados que ganham menos que seu gerente:

```
SELECT E.ENAME EMP_NAME,
       E.SAL EMP_SAL,
       M.ENAME MGR_GSAL
FROM   EMP E, EMP M
WHERE  E.MGR = M.EMPNO
AND    E.SAL < MSAL;
```

| EMP_NAME | EMP_SAL | MGR_NAME | MGR_SAL |
|----------|---------|----------|---------|
| ----- | ----- | ----- | ----- |
| ALLEN | 1600 | BLAKE | 2850 |
| WARD | 1250 | BLAKE | 2850 |
| MARTIN | 1250 | BLAKE | 2850 |

| | | | |
|--------|------|-------|------|
| TURNER | 1500 | BLAKE | 2850 |
| JAMES | 950 | BLAKE | 2850 |
| MILLER | 1300 | CLARK | 2450 |
| ADAMS | 1100 | SCOTT | 3000 |
| JONES | 2975 | KING | 5000 |
| BLAKE | 2850 | KING | 5000 |
| CLARK | 2450 | KING | 5000 |
| SMITH | 800 | FORD | 3000 |

Note que a cláusula FROM refere-se para EMP duas vezes, e portanto EMP tem que ter um sinônimo para cada caso - E e M. Atribuir sinônimo para uma tabela significa por exemplo E significa empregados e M significa gerentes. Quando um empregado é gerente o número de seu gerente é o seu mesmo.

Operadores de Conjunto

Durante a introdução, o conceito de Operadores de Conjunto foram discutidos. Assim distante o curso tem falado sobre restrições, ligações, projeções e produtos. INTERSECT, UNION e MINUS são falados aqui. UNION, INTERSECT e MINUS são geralmente usados para diferenciar tabelas. Eles combinam resultados de duas ou mais declarações de SELECT em um resultado. A pesquisa deve consistir de duas ou mais declarações do SQL ligadas por um ou mais operador de conjunto. Operadores de conjunto são geralmente chamados de Ligações Verticais, porque a ligação não é de acordo para linhas entre tabelas, mas colunas.

Nos seguintes três exemplos, as pesquisas são as mesmas, mas o operador é diferente em cada caso é produzido um resultado de pesquisa diferente.

UNION

Para retornar as linhas distintas de cada uma das pesquisas, faça:

```
SELECT JOB
FROM EMP
WHERE DEPTNO = 10
UNION
SELECT JOB
FROM EMP
WHERE DEPTNO = 30;
```

```
JOB
-----
CLERK
MANAGER
PRESIDENT
SALESMAN
```

Nota: UNION ALL retorna todas as linhas das duas pesquisas independente se houver mais de uma linha igual

INTERSECT

Para retornar somente as linhas que estão em ambas as pesquisas, faça:

```
SELECT JOB
FROM EMP
WHERE DEPTNO = 10
INTERSECT
SELECT JOB
FROM EMP
WHERE DEPTNO = 30;

JOB
-----
CLERK
MANAGER
```

MINUS

Para retornar todas as linhas que tem na primeira pesquisa que não tem na segunda, faça:

```
SELECT JOB
FROM EMP
WHERE DEPTNO = 10
MINUS
SELECT JOB
FROM EMP
WHERE DEPTNO = 30;

JOB
-----
PRESIDENT
```

É possível construir pesquisas com vários operadores de conjuntos. Se vários operadores de conjuntos forem usados, a ordem de execução para declaração SQL é de cima para baixo. Parênteses podem ser usados para produzir uma alternativa ordem de execução.

ORDER BY

ORDER BY tem que ser único em uma pesquisa que emprega operadores de conjunto. Se usado, o ORDER BY deve ser colocado no final da pesquisa. Então, porque você deve selecionar diferentes colunas em cada SELECT você não pode usar o nome das colunas na cláusula ORDER BY. Entretanto as colunas no ORDER BY devem ser referidas pelo sua relativa posição na lista do SELECT.

```
SELECT EMPNO, ENAME, SAL
FROM EMP
UNION
```



```
SELECT ID, NAME, SALARY  
FROM EMP_HISTORY  
ORDER BY 2;
```

Note que no ORDER BY um numeral (2) é usado para representar a posição do ENAME coluna da lista do SELECT. Esse significa que as linhas serão retornadas em ordem ascendente nome do empregado.

Regras Quando Usar Operadores de Conjuntos.

1. Declarações do SELECT devem selecionar o mesmo número de colunas.
2. As colunas correspondentes devem ser do mesmo tipo.
3. Linhas duplicadas serão automaticamente eliminadas.,(DISTINCT não pode ser usado).
4. O nome das colunas da primeira pesquisa será os que aparecerão no resultado.
5. A cláusula ORDER BY aparecerá no final da declaração.
6. Na cláusula ORDER BY será colocada apenas as posições das colunas
7. Operadores de Conjuntos podem ser usados em sub-pesquisas
8. A declaração das pesquisas serão executadas de cima para baixo.
9. Vários operadores de conjuntos podem ser usados juntos com parênteses para alterar a seqüência de execução.

Exercício 6 - Outros Métodos de Ligações

Esses exercícios tem a função para você ter a oportunidade de descobrir métodos alternativos de ligações e algumas experiência com operadores de conjuntos.

1. Mostrar a seguinte informação

| EMPNO | ENAME | DNAME | LOC |
|-------|--------|------------|----------|
| ----- | ----- | ----- | ----- |
| 7782 | CLARK | ACCOUNTING | NEW YORK |
| 7839 | KING | ACCOUNTING | NEW YORK |
| 7934 | MILLER | ACCOUNTING | NEW YORK |
| 7876 | ADAMS | RESEARCH | DALLAS |
| 7902 | FORD | RESEARCH | DALLAS |
| 7566 | JONES | RESEARCH | DALLAS |
| 7788 | SCOTT | RESEARCH | DALLAS |
| 7369 | SMITH | RESEARCH | DALLAS |
| 7499 | ALLEN | SALES | CHICAGO |
| 7698 | BLAKE | SALES | CHICAGO |
| 7900 | JAMES | SALES | CHICAGO |
| 7654 | MARTIN | SALES | CHICAGO |
| 7844 | TURNER | SALES | CHICAGO |
| 7521 | WARD | SALES | CHICAGO |

2. Mostrar os departamentos que não tem empregados

| DEPTNO | DNAME |
|--------|------------|
| ----- | ----- |
| 40 | OPERATIONS |

3. Listar todos os empregados com seus nomes e números adiante seus gerentes com nome e número:

| EMPNO | ENAME | MGRNO | MGR_NAMR |
|-------|--------|-------|----------|
| ----- | ----- | ----- | ----- |
| 7782 | CLARK | 7839 | KING |
| 7934 | MILLER | 7782 | CLARK |
| 7876 | ADAMS | 7788 | SCOTT |
| 7902 | FORD | 7566 | JONES |
| 7566 | JONES | 7839 | KING |
| 7788 | SCOTT | 7566 | JONES |
| 7369 | SMITH | 7902 | FORD |
| 7499 | ALLEN | 7698 | BLAKE |
| 7698 | BLAKE | 7839 | KING |
| 7900 | JAMES | 7698 | BLAKE |
| 7654 | MARTIN | 7698 | BLAKE |
| 7844 | TURNER | 7698 | BLAKE |
| 7521 | WARD | 7698 | BLAKE |

4. Modifique a solução da questão 2 para mostrar KING que não tem gerente:

| EMPNO | ENAME | MGRNO | MGR_NAMR |
|-------|--------|-------|----------|
| ----- | ----- | ----- | ----- |
| 7839 | KING | | |
| 7782 | CLARK | 7839 | KING |
| 7934 | MILLER | 7782 | CLARK |
| 7876 | ADAMS | 7788 | SCOTT |
| 7902 | FORD | 7566 | JONES |
| 7566 | JONES | 7839 | KING |
| 7788 | SCOTT | 7566 | JONES |
| 7369 | SMITH | 7902 | FORD |
| 7499 | ALLEN | 7698 | BLAKE |

| | | | |
|------|--------|------|-------|
| 7698 | BLAKE | 7839 | KING |
| 7900 | JAMES | 7698 | BLAKE |
| 7654 | MARTIN | 7698 | BLAKE |
| 7844 | TURNER | 7698 | BLAKE |
| 7521 | WARD | 7698 | BLAKE |

5. Encontre o cargo que era preenchido no primeiro semestre de 1983, e o mesmo cargo que era preenchido durante o mesmo período em 1984.

JOB

CLERK

6. Encontre todos os empregados que ligaram-se a companhia antes de seu gerente.

| ENAME | HIREDATE | MGR | HIREDATE |
|--------|-----------|-------|-----------|
| SMITH | 13-JUN-83 | FORD | 05-DEC-83 |
| ALLEN | 15-AUG-83 | BLAKE | 11-JUN-84 |
| WARD | 26-MAR-84 | BLAKE | 11-JUN-84 |
| JONES | 31-OCT-83 | KING | 09-JUL-84 |
| MARTIN | 05-DEC-83 | BLAKE | 11-JUN-84 |
| BLAKE | 11-JUN-84 | KING | 09-JUL-84 |
| CLARK | 14-MAY-84 | KING | 09-JUL-84 |
| TURNER | 04-JUN-84 | BLAKE | 11-JUN-84 |
| MILLER | 21-NOV-83 | CLARK | 14-MAY-84 |

7. Encontre outra método de pesquisa para questão 2:

| DEPTNO | DNAME |
|--------|------------|
| 40 | OPERATIONS |

Sub-pesquisas (Subqueries)

Nessa Unidade nós devemos falar sobre mais avançados traços da declaração SELECT, isto é:

- pesquisas contendo na clausula WHERE ou HAVING de outra declaração SQL

Aninhamento de Sub-pesquisas.

Uma sub-pesquisa é uma declaração SELECT que é aninhada com outra declaração SELECT e a qual retorna resultados intermediários.

Por exemplo:

```
SELECT coluna1, coluna2, ...
FROM tabela
WHERE coluna =
      (SELECT coluna
```

```
FROM   tabela
WHERE  condição)
```

A sub-pesquisa é geralmente referida como SUB-SELECT ou SELECT interno; ele geralmente executa primeiro e a saída é usada para completar a condição da pesquisa principal ou outra pesquisa. Usando sub-pesquisas permite um desenvolvimento para construções de potentes comandos de saída fáceis. O aninhamento de sub-pesquisas pode ser usado quando você precisa selecionar linhas de uma tabela com uma condição que depende de dados na mesma tabela.

Sub-pesquisas de Linha Única

Para encontrar o empregado que ganha o mínimo salário na companhia (o mínimo salário é uma quantidade desconhecida), dois passos devem ser seguidos:

1. Encontrar o salário mínimo:

```
SELECT MIN(SAL) FROM EMP;
```

| |
|----------|
| MIN(SAL) |
| ----- |
| 800 |

2. Encontrar o empregado que ganha o salário mínimo:

```
SELECT ENAME, JOB, SAL
FROM EMP
WHERE SAL = (menor salário o qual é desconhecido)
```

Nós podemos combinar os dois comandos como uma sub-pesquisa aninhada:

```
SELECT ENAME, JOB, SAL
FROM EMP
WHERE SAL = SELECT MIN(SAL) FROM EMP;
```

| | | |
|-------|-------|-------|
| ENAME | JOB | SAL |
| ----- | ----- | ----- |
| SMITH | CLERK | 800 |

Como são processadas as Sub-pesquisas Aninhadas?

Uma declaração SELECT pode ser considerada como uma pesquisa em bloco. O exemplo acima consiste de duas pesquisas em bloco - a principal pesquisa e a pesquisa interna. A interna declaração SELECT é executada primeiro, produzindo um resultado : 800. A principal pesquisa em bloco está então processando e usa o valor retornado pela pesquisa interna para completar a condição procurada. Na essência, a principal pesquisa finaliza-se olhando como isso:

```
SELECT ENAME, SAL, DEPTNO
```

```
FROM EMP
WHERE SAL = 800;
```

No exemplo acima, o 800 é um valor único. A sub-pesquisa que retorna o valor 800 é chamada de sub-pesquisa de linha única. Quando uma sub-pesquisa retorna uma única linha: uma linha ou operador lógico deve ser usado. Por exemplo: =, <, >, <=, etc.

Para encontrar todos os empregados que tem o mesmo cargo como BLAKE nós fazemos:

```
SELECT ENAME, JOB
FROM EMP
WHERE JOB = (SELECT JOB
              FROM EMP
              WHERE ENAME = 'BLAKE');
```

| ENAME | JOB |
|-------|---------|
| JONES | MANAGER |
| BLAKE | MANAGER |
| CLARK | MANAGER |

A pesquisa interna retorna o cargo de BLAKE o qual é usado na condição WHERE da pesquisa principal.

Sub-pesquisas que Retorna mais de Uma Linha

A seguinte pesquisa atende para encontrar os empregados que ganham o menor salário nos departamentos.

```
SELECT ENAME, SAL, DEPTNO FROM EMP
WHERE SAL IN (SELECT MIN(SAL)
              FROM EMP
              GROUP BY DEPTNO);
```

| ENAME | SAL | DEPTNO |
|--------|------|--------|
| SMITH | 800 | 20 |
| JAMES | 950 | 30 |
| MILLER | 1300 | 10 |

Note que a pesquisa interna tem a clausula GROUP BY. Isso significa que ele pode retornar mais que um valor. Nós precisamos para usar múltiplas linhas de um operador de comparação. Neste caso o operador IN deve ser usado porque especifica uma lista de valores. O resultado obtido não mostra o departamento em que o qualificado empregado trabalha. No entanto, porque nós estamos comparando unicamente o valor do salário, a pesquisa interna pode retornar um valor simples porque ela combina o menor salário para um departamento, não necessariamente um empregado para cada departamento. A pesquisa deve ser rescrita em ordem para possuir a combinação de salários de empregados e o número do departamento com os salários mínimos e os números de departamentos:

Comparando mais de um valor

A seguinte pesquisa encontrará aqueles empregados que ganham o menor salário no seu respectivo departamento:

```
SELECT ENAME, SAL, DEPTNO
FROM EMP
WHERE (SAL,DEPTNO) IN (SELECT MIN(SAL), DEPTNO
                       FROM EMP
                       GROUP BY DEPTNO);
```

| ENAME | SAL | DEPTNO |
|--------|------|--------|
| SMITH | 800 | 20 |
| JAMES | 950 | 30 |
| MILLER | 1300 | 10 |

A pesquisa acima compara uma parte da coluna. Note que as colunas da esquerda da condição procurada estão entre parênteses e que as colunas estão separadas com uma vírgula. Colunas listadas na clausula SELECT da sub-pesquisa deve estar na mesma ordem das colunas listadas na clausula WHERE da outra pesquisa e o mesmo tipo de colunas.

Por exemplo:

```
....WHERE (numcoluna, charcoluna) =
(SELECT datacoluna, numcoluna, charcoluna...
```

é ilegal.

Erros Encontrados

Quando uma sub-pesquisa retorna mais que uma linha e um operador de linha única é usado, SQL*Plus mostra o seguinte mensagem de erro:

```
SELECT ENAME, SAL, DEPTNO
FROM EMP
WHERE SAL = (SELECT MIN(SAL)
             FROM EMP
             GROUP BY DEPTNO);
```

```
ERROR: ORA-1427: single-row subquery returns more than one row
(Sub-pesquisa de linha única retornou mais de uma linha)
```

Se a pesquisa interna não retornar linhas, você tem o erro:

```
SELECT ENAME, JOB
FROM EMP
WHERE JOB = (SELECT JOB
             FROM EMP
             WHERE ENAME = 'SMYTHE');
```

```
ERROR: ORA-1426: single-row subquery returns no rows
(Sub-pesquisa de linha única não retornou nenhuma linha)
```

Operadores ANY ou ALL

Os operadores ANY ou ALL devem ser usados para sub-pesquisas que retornam mais de uma linha. Eles são usados na cláusula WHERE ou HAVING em conjunto com os operadores lógicos. (=, <>, <, >, >=, <=).

ANY compara um valor para cada valor retornado em uma sub-pesquisa.

Para mostrar os empregados que ganham mais que o menor salário no departamento 30, faça:

```
SELECT ENAME, SAL, JOB, DEPTNO
FROM EMP
WHERE SAL > ANY (SELECT DISTINCT SAL
                 FROM EMP
                 WHERE DEPTNO = 30)
ORDER BY SAL DESC;
```

| ENAME | SAL | JOB | DEPTNO |
|--------|------|-----------|--------|
| KING | 5000 | PRESIDENT | 10 |
| SCOTT | 3000 | ANALYST | 20 |
| FORD | 3000 | ANALYST | 20 |
| JONES | 2975 | MANAGER | 20 |
| BLAKE | 2850 | MANAGER | 30 |
| CLARK | 2450 | MANAGER | 10 |
| ALLEN | 1600 | SALESMAN | 30 |
| TURNER | 1500 | SALESMAN | 30 |
| MILLER | 1300 | CLERK | 10 |
| WARD | 1250 | SALESMAN | 30 |
| MARTIN | 1250 | SALESMAN | 30 |
| ADAMS | 1100 | CLERK | 20 |

O menor salário do departamento 30 é 950 (James). A principal pesquisa tem que retornar os empregados que ganham o salário maior que o menor salário no departamento 30. ‘<ANY’ é equivalente ao IN. Quando usamos ANY, a palavra chave DISTINCT é usada para prevenir a seleção de linhas ocupadas. ALL compara um valor todos os valores retornados em uma sub-pesquisa.

A seguinte pesquisa encontra os empregados que ganham mais que todos os empregados no departamento 30.

```
SELECT ENAME, SAL, JOB, DEPTNO
FROM EMP
WHERE SAL > ALL (SELECT DISTINCT SAL
                 FROM EMP
                 WHERE DEPTNO = 30)
ORDER BY SAL DESC;
```

| ENAME | SAL | JOB | DEPTNO |
|-------|------|-----------|--------|
| KING | 5000 | PRESIDENT | 10 |
| SCOTT | 3000 | ANALYST | 20 |
| FORD | 3000 | ANALYST | 20 |
| JONES | 2975 | MANAGER | 20 |

O maior salário no departamento 30 é 2850 (Blake), a pesquisa tem que retornar aqueles empregados que ganham mais que 2850. Existe salário maior que o maior do departamento 30. O operador NOT podem ser usado com IN, ANY ou ALL.

Clausula HAVING com Sub-pesquisas aninhadas.

Sub-pesquisa aninhadas então podem ser usadas na clausula HAVING (Lembre-se que WHERE refere-se para linha única e HAVING a grupos de linhas especificadas na clausula GROUP BY.

Por exemplo, para mostrar os departamentos que tenham a média salarial maior que a do departamento 30, faça:

```
SELECT DEPTNO, AVG(SAL) FROM EMP
HAVING AVG(SAL) > (SELECT AVG(SAL)
                   FROM EMP
                   WHERE DEPTNO = 30)
GROUP BY DEPTNO;
```

| DEPTNO | AVG(SAL) |
|--------|------------|
| 10 | 2916.66667 |
| 20 | 2175 |

Para construir uma pesquisa que encontre o cargo com maior média salarial, faça:

```
SELECT JOB, AVG(SAL)
FROM EMP
GROUP BY JOB
HAVING AVG(SAL) = (SELECT MAX(AVG(SAL))
                  FROM EMP
                  GROUP BY JOB);
```

| JOB | AVG(SAL) |
|-----------|----------|
| PRESIDENT | 5000 |

A primeira pesquisa interna encontra as médias salariais de cada diferente grupo de cargo, e a função MAX seleciona a maior média salarial. Aquele valor (5000) é usada na clausula HAVING. A clausula GROUP BY na pesquisa principal é necessária porque a declaração SELECT possui uma coluna que não é um item de grupo.

Ordenando Dados com Sub-pesquisas

Você **não** deve ter uma clausula ORDER BY na sub-pesquisa. A regra restante que você pode ter unicamente uma clausula ORDER BY para uma declaração SELECT e, se especificar, deve ser a última clausula no comando SELECT.

Sub-pesquisas Aninhadas

Sub-pesquisas devem ser aninhadas (usadas com outras sub-pesquisas):

Mostrar o nome, cargo e admissão para os empregados que o salário é maior que o maior salário no departamento 'SALES'.

```
SELECT ENAME, JOB, HIREDATE, SAL
FROM EMP
WHERE SAL > (SELECT MAX(SAL)
              FROM EMP
              WHERE DEPTNO = (SELECT DEPTNO
                              FROM DEPT
                              WHERE DNAME = 'SALES'));
```

| ENAME | JOB | HIREDATE | SAL |
|-------|-----------|-----------|----------|
| JONES | MANAGER | 31-OCT-83 | 2,975.00 |
| SCOTT | ANALYST | 05-MAR-84 | 3,000.00 |
| KING | PRESIDENT | 09-JUL-84 | 5,000.00 |
| FORD | ANALYST | 05-DEC-83 | 3,000.00 |

Limites de Alinhamento

Não existe limite para níveis de aninhamento para sub-pesquisas.

Diretriz

- A pesquisa interna precisa ser incluída entre parênteses. e precisa estar do lado direito da condição.
- Uma sub-pesquisa não deve ter uma clausula ORDER BY.
- As várias colunas na lista do SELECT da pesquisa interna deve ser na mesma ordem como as colunas aparecem na clausula de condição da pesquisa principal.
- Sub-pesquisas são sempre executadas da mais profunda até a menos profunda, a menos que sejam correlatas.
- Operadores Lógicos e SQL devem ser usados como ANY e ALL.
- Sub-pesquisas podem:
 - Retornar uma ou mais linhas.
 - Retornar uma ou mais colunas
 - Usar o GROUP BY e funções de grupo
 - Ser usada em vários predicados AND ou OR da mesma pesquisa externa
 - Ligar Tabelas
 - Retirar de diferentes tabelas que a externa
 - Aparecer em declarações de SELECT, UPDATE, DELETE, INSERT, CREATE TABLE.
 - Correlacionar com uma pesquisa externa.
 - Usar operadores de Conjunto.

Sub-pesquisas Correlatas

Uma Sub-pesquisa Correlata é uma sub-pesquisa aninhada que é executada uma vez para cada linha candidata considerada pela pesquisa principal e que na execução usa um valor de coluna da pesquisa externa. Isso origina a sub-pesquisa correlata para ser processada em uma diferente forma normal de Aninhamento de Sub-pesquisas. Uma Sub-pesquisa Correlata é identificada pelo uso de colunas da pesquisa externa em sua condição. Com uma Sub-pesquisa aninhada normal, a seleção interna é executada primeiro e ela executa uma vez, retornando valores para serem usados na pesquisa principal. Uma Sub-pesquisa Correlata, é um outro modo, executa uma vez para cada linha candidata considerada na pesquisa externa. A pesquisa interna é dirigida pela externa.

Passos para executar Sub-pesquisas Correlatas.

1. Pegar linhas candidatas (trazer na pesquisa externa)
2. Executar pesquisa interna usando valores da linha candidata.
3. Usar valores retornados da pesquisa interna para qualificar ou desqualificar candidatas.
4. Repetir até não haver mais linhas candidatas.

Embora as Sub-pesquisas correlatas executam repetitivamente, uma vez para cada linha da pesquisa principal, não existem sugestões de que elas são menos eficiente que sub-pesquisas não correlatas. Nós podemos usar Sub-pesquisas Correlatas para encontrar empregados que ganham um salário maior que a média salarial para seus departamentos:

```
SELECT EMPNO, ENAME, SAL, DEPTNO
FROM EMP E
WHERE SAL > (SELECT AVG(SAL)
              FROM EMP
              WHERE DEPTNO = E.DEPTNO)
ORDER BY DEPTNO;
```

| EMPNO | ENAME | SAL | DEPTNO |
|-------|-------|------|--------|
| 7839 | KING | 5000 | 10 |
| 7566 | JONES | 2975 | 20 |
| 7788 | SCOTT | 3000 | 20 |
| 7902 | FORD | 3000 | 20 |
| 7499 | ALLEN | 1600 | 30 |
| 7698 | BLAKE | 2850 | 30 |

Nós podemos ver imediatamente que esse é uma pesquisa correlata então nós temos de usar uma coluna do SELECT externo no WHERE do SELECT interno. Note que o sinônimo é necessário para evitar ambigüidade nos nomes das colunas.

Vamos nós analisar o exemplo acima usando a tabela EMP:

A Pesquisa Principal:

1. Seleciona primeiro a linha candidata - Smith no departamento 20 ganhando 800.
2. EMP da clausula FROM tem um sinônimo E o qual qualifica a coluna DEPTNO referenciada no WHERE da pesquisa interna.

3. Clausula WHERE compara 800 como valor retornado pela pesquisa interna.

A Pesquisa Interna

4. Calculo AVG(SAL) para empregados do departamento.
5. O valor da coluna E.DEPTNO é o valor passado pela pesquisa externa.
6. AVG(SAL) para Smith do departamento 20 é 2175.
7. A linha candidata não encontra a condição, por isso é descartada.
8. Repetir o passo 1 para cada linha candidata.

A seleção das linhas candidatas continua com aquelas que encontrar as condições que aparecem no resultado da pesquisa. Lembre-se uma Sub-pesquisa Correlata é originada por uma coluna de uma tabela ou tabela sinônima na clausula WHERE que refere-se ao valor da coluna em cada linha candidata da seleção externa. Então Sub-pesquisa Correlata executa repetitivamente para cada linha candidata da pesquisa principal.

O comando UPDATE pode conter Sub-pesquisas Correlatas.

```
UPDATE EMP E
SET (SAL, COMM) = (SELECT AVG(SAL)*1.1, AVG(COMM)
                   FROM EMP
                   WHERE DEPTNO = E.DEPTNO)
WHERE HIREDATE = '11-JUN-85';
```

O comando UPDATE será estudado mais adiante.

Operadores

Quando você está aninhando declarações do SELECT os operadores lógicos são todos validos como também ANY e ALL. Nas demais o operador EXISTS precisa ser usado.

Operador EXISTS

O operador EXISTS é freqüentemente usado com Sub-pesquisas correlatas. Ele testa quando um valor existe. (NOT EXISTS garante que não existe). Se o valor existir será retornado Verdadeiro, se não existir será retornado Falso.

Para encontrar os empregados que tem no mínimo uma pessoa subordinada a ele, faça:

```
SELECT EMPNO, ENAME, JOB, DEPTNO
FROM EMP E
WHERE EXISTS (SELECT EMPNO
              FROM EMP
              WHERE EMP.MGR = E.EMPNO)
ORDER BY EMPNO;
```

| EMPNO | ENAME | JOB | DEPTNO |
|-------|-------|---------|--------|
| 7566 | JONES | MANAGER | 20 |

| | | | |
|------|-------|-----------|----|
| 7698 | BLAKE | MANAGER | 30 |
| 7782 | CLARK | MANAGER | 10 |
| 7788 | SCOTT | ANALYST | 20 |
| 7839 | KING | PRESIDENT | 10 |
| 7902 | FORD | ANALYST | 20 |

Encontrar os empregados que o departamento não é o da tabela DEPT:

```
SELECT, ENAME, DEPTNO
FROM EMP
WHERE NOT EXISTS (SELECT DEPTNO
                   FROM DEPT
                   WHERE DEPT.DEPTNO = EMP.DEPTNO);
```

no records selected

Outro caminho para encontrar o departamento que não tem nenhum empregado é:

```
SELECT DEPTNO, DNAME
FROM DEPT D
WHERE NOT EXISTS (SELECT 'X'
                   FROM EMP E
                   WHERE E.DEPTNO = D.DEPTNO);
```

| DEPTNO | DNAME |
|--------|------------|
| ----- | ----- |
| 40 | OPERATIONS |

Note que o SELECT interno não precisa retornar um valor específico, unicamente um literal para ser selecionado.

Por que Usar uma Sub-pesquisa Correlata?

A Sub-pesquisa Correlata é um caminho de ler todas as linhas na tabela, e comparando os valores em cada linha retornada. Ela é usada quando uma sub-pesquisa precisa retornar um diferente resultado ou conjuntos de resultados para cada linha candidata considerada na pesquisa principal. Em outras palavras, uma sub-pesquisa correlata é usada para responder questões que as respostas dependem de valores em cada linha da pesquisa parente.

O SELECT interno normalmente é executado uma vez para cada linha candidata.

Considerações de Eficiência.

Nós temos agora examinadas dois tipos de sub-pesquisas. Ela é importante referência que a sub-pesquisa correlata (com EXISTS) pode ser o mais eficiente caminho de agilizar algumas pesquisas.

Performance depende do uso dos índices, o número de linhas retornadas pela pesquisa, o tamanho da tabela e se tabelas temporárias são requeridas para avaliar resultados temporários. As tabelas temporárias geradas pelo ORACLE não são indexadas, e essa pode levar para o degradamento na performance para sub-pesquisas usando IN, ANY e ALL.

Exercício 7 - Sub-pesquisas

Esses exercícios permite a você a escrever complexas pesquisas usando seleções aninhadas e seleções correlatas.

1. Encontrar os empregados que ganham o maior salário em cada cargo e ordenar o salário da forma descendente.

| JOB | ENAME | SAL |
|-----------|--------|----------|
| PRESIDENT | KING | 5,000.00 |
| ANALYST | SCOTT | 3,000.00 |
| ANALYST | FORD | 3,000.00 |
| MANAGER | JONES | 2,975.00 |
| SALESMAN | ALLEN | 1,600.00 |
| CLERK | MILLER | 1,300.00 |

2. Encontrar os empregados que ganham o mínimo salário para seu cargo. Mostrar o resultado em ordem ascendente de salário.

| ENAME | JOB | SAL |
|--------|-----------|----------|
| SMITH | CLERK | 800.00 |
| WARD | SALESMAN | 1,250.00 |
| MARTIN | SALESMAN | 1,250.00 |
| CLARK | MANAGER | 2,450.00 |
| SCOTT | ANALYST | 3,000.00 |
| FORD | ANALYST | 3,000.00 |
| KING | PRESIDENT | 5,000.00 |

3. Encontrar o mais recente admitido empregado em cada departamento. Ordenado por admissão.

| DEPTNO | ENAME | HIREDATE |
|--------|-------|-----------|
| 20 | ADAMS | 04-JUN-84 |
| 10 | KING | 09-JUL-84 |
| 30 | JAMES | 23-JUL-84 |

4. Mostre os seguintes detalhes para qualquer empregado que ganhe um salário maior que a média para seu departamento. Ordenar pelo número de departamento.

| ENAME | SAL | DEPTNO |
|-------|----------|--------|
| KING | 5,000.00 | 10 |
| JONES | 2,975.00 | 20 |
| SCOTT | 3,000.00 | 20 |
| FORD | 3,000.00 | 20 |
| ALLEN | 1,600.00 | 30 |
| BLAKE | 2,850.00 | 30 |

5. Lista todos os departamentos onde não existem empregados.
(Usando dessa vez um sub-pesquisa).

| DEPTNO | DNAME |
|--------|-------|
|--------|-------|

```

-----
40                OPERATIONS

```

6. Mostrar as seguintes informações para o departamento com o maior remuneração anual faturada.

```

DEPTNO      COMPENSATION
-----
20          130500

```

7. Quem são os três maiores salários da companhia? Mostrar nome e salário.

```

ENAME      SAL
-----
SCOTT      3,000.00
KING       5,000.00
FORD       3,000.00

```

8. Em qual ano de maior ligações de pessoas a companhia? Mostrar o ano e número de empregados.

```

YEAR      NUMBER_OF_EMPS
-----
1984      8

```

9. Modificar a questão 4 para então mostrar a média salarial figurada para o departamento.

```

ENAME      SAL      DEPTNO      DEPAVG
-----
KING       5,000.00      10      1566.66667
JONES      2,975.00      20      1566.66667
SCOTT      3,000.00      20      2175
FORD       3,000.00      20      2175
ALLEN      1,600.00      30      2175
BLAKE      2,850.00      30      2916.66667

```

10. Escrever uma pesquisa para mostrar um * na linha do mais recente empregado admitido. Mostrar o nome, admissão e mostrar um * no (MAXDATE).

```

ENAME      HIREDATE      MAXDATE
-----
SMITH      13-JUN-83
ALLEN      15-AUG-83
WARD       26-MAR-84
JONES      31-OCT-83
MARTIN     05-DEC-83
BLAKE      11-JUN-84
CLARK      14-MAY-84
SCOTT      05-MAR-84
KING       09-JUL-84
TURNER     04-JUN-84
ADAMS      04-JUN-84
JAMES      23-JUL-84      *
FORD       05-DEC-83
MILLER     21-NOV-83

```

Linguagem de Manipulação dos Dados

Essa Unidade explica como fazer modificações para linhas na tabela, adicionar linhas ou apagá-las. A concepção de transação é introduzida. Consistência de leitura também é falada.

INSERT

O comando INSERT é usada para adicionar linhas em uma tabela.

A sintaxe do comando INSERT é:

```
INSERT INTO nome_tabela (coluna, coluna, ...)
VALUES (valor, valor, ...);
```

Ele possibilita inserir novas linhas com valores em cada coluna. Ele recomenda que a colunas listadas sejam sempre especificadas. Se a lista não é especificada seu programa terá que ser modificado quando a definição da tabela for alterada.

Para inserir um novo departamento, faça:

```
INSERT INTO DEPT (DEPTNO, DNAME, LOC)
VALUES (50, 'MARKETING', 'SAN JOSE');
```

Note que esse comando adiciona unicamente uma linha de uma vez na tabela.

Para entrar com um novo departamento omitindo o nome, a coluna deve ser omitida.

```
INSERT INTO DEPT (DEPTNO, LOC)
VALUES (50, 'SAN JOSE');
```

Alternativamente se o nome de departamento é desconhecido, um NULL pode ser especificado.

```
INSERT INTO DEPT (DEPTNO, DNAME, LOC)
VALUES (50, NULL, 'SAN JOSE');
```

Valores ALFANUMÉRICOS e DATAS devem ser incluídos entre aspas simples (').

Usando Variáveis Substituíveis para linhas inseridas.

Como previamente mencionamos, INSERT é uma linha por comando. Usando Variáveis Substituíveis ele pode ser reutilizado:

```
INSERT INTO DEPT (DEPTNO, DNAME, LOC)
VALUES (&D_NUMBER, '&D_NAME', '&LOCATION');
```

Quando o comando é executado, os valores são solicitados todas as vezes.

Inserindo Datas e informações de Horas.

Quando inserimos um valor data, o formato DD-MON-YY é regularmente usado. Com esse formato o século padrão é o 19. A data contém informações de hora, a qual se não especificada o padrão é meia noite.(00:00:00).

Se a data precisar ser entrada em outro século e especificar a hora é requerido usar a função TO_DATE:

```
INSERT INTO EMP
      (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
VALUES (7658,
        'MASON',
        'ANALYST',
        7566,
        TO_DATE('24/06/2084 9:30', 'DD/MM/YYYY HH:MI'),
        3000,
        NULL,
        20);
```

Copiando linhas de Outra Tabela

```
INSERT INTO tabela (coluna, coluna, ...)
SELECT coluna, coluna, ...
FROM tabela(s);
```

Essa forma de INSERT permite a você inserir várias linhas em uma tabela onde os valores são derivados de conteúdo existente em outras tabelas no Banco de Dados. Para copiar todas as informações sobre o departamento 10 na tabela D10HISTORY, faça:

```
INSERT INTO D10HISTORY
      (EMPNO, ENAME, SAL, JOB, HIREDATE)
SELECT EMPNO, ENAME, SAL, JOB, HIREDATE
FROM EMP
WHERE DEPTNO = 10;
```

Note que a palavra chave VALUES não é usada aqui.

UPDATE

A declaração UPDATE permite você modificar valores nas linhas em uma tabela:

```
UPDATE tabela (sinônimo)
SET coluna,(coluna,...) = (expressão, sub-pesquisa)
(WHERE condição)
```

Por exemplo. para alterar a linha do Scott faça:

```
UPDATE EMP
SET JOB='SALESMAN',
    HIREDATE=SYSDATE,
```



```
SAL=SAL*1.1
WHERE ENAME = 'SCOTT';
```

1 record updated.

Se a clausula WHERE for omitida, todas as linhas da tabela serão alteradas.

É possível usar sub-pesquisas aninhadas e correlatas na declaração UPDATE.

Suponhamos que você tenha uma nova tabela do comissões que deve ser distribuída em certos empregados. Por exemplo a tabela de comissão abaixo.

| EMPNO | COMM |
|-------|------|
| 7499 | 666 |
| 7654 | 758 |

As modificações listadas na tabela COMMISSION pode ser aplicada na tabela EMP usando uma sub-pesquisa correlata e uma sub-pesquisa aninhada como mostramos abaixo.

Exemplo 1:

```
UPDATE EMP
SET COMM = (SELECT COMM FROM COMMISSION C
            WHERE C.EMPNO = EMP.EMPNO)
WHERE EMPNO IN (SELECT EMPNO FROM COMMISSION);
```

2 records updated.

A tabela COMMISSION pode conter mais de uma linha para cada empregado abaixo:

| EMPNO | COMM |
|-------|--------|
| 7499 | 666.00 |
| 7521 | 500.00 |
| 7521 | 500.00 |
| 7654 | 758.00 |

Você quer trocar o valor da comissão da tabela EMP com o total das comissões para cada empregado lista na tabela COMMISSION.

Para fazer isso, use o seguinte SQL:

Exemplo 2:

```
UPDATE EMP
SET COMM = (SELECT SUM(COMM) FROM COMMISSION C
            WHERE C.EMPNO = EMP.EMPNO)
WHERE EMPNO IN (SELECT EMPNO FROM COMMISSION);
```

3 records updated.

A tabela EMP agora reflete as modificações das comissões:

| EMPNO | COMM |
|-------|----------|
| 7499 | 666.00 |
| 7521 | 1,000.00 |

7654 758.00

Outra possibilidade é que você queira adicionar o valor da comissão existente na tabela EMP com o valor da comissão da tabela COMMISSION. Exemplo 3 faz isso:

Exemplo 3:

```
UPDATE EMP
SET COMM = (SELECT SUM(COMM) + EMP.COMM
            FROM COMMISSION C
            WHERE C.EMPNO = EMP.EMPNO)
WHERE EMPNO IN (SELECT EMPNO FROM COMMISSION);
```

DELETE

O comando DELETE permite você remover uma ou mais linhas de uma tabela.

```
DELETE FROM tabela
```

(WHERE condição);

Para apagar todas as informações referentes ao departamento 10 da tabela EMP faça:

```
DELETE FROM EMP
WHERE DEPTNO =10;
```

Se a clausula WHERE for omitida todas linhas da tabela EMP serão excluídas.

COMMIT E ROLLBACK

TRANSAÇÕES

| -----TRANSAÇÕES----- | | | | | |
|----------------------|--------|--------|--------|--------|-------|
| INSERT | DELETE | INSERT | UPDATE | UPDATE | TEMPO |
| COMMIT | COMMIT | COMMIT | | | |

ou

ROLLBACK

Até que as modificações sejam confirmadas:

- você pode ver então na pesquisa
- outros usuários não podem ver
- você deve descartar (ROLLBACK)
- você deve ROLLBACK em um SAVEPOINT

Processando Transações

O que é uma Transação?

Uma transação é uma operação contra o Banco de Dados a qual compromete uma série de modificações de uma ou mais tabelas.

Existem duas classes de Transações. Transações DML a qual pode consistir de qualquer número de declarações DML a qual o ORACLE trata como única entidade ou lógica unidade de trabalho, e transações DDL a qual pode unicamente consistir de uma declaração DDL.

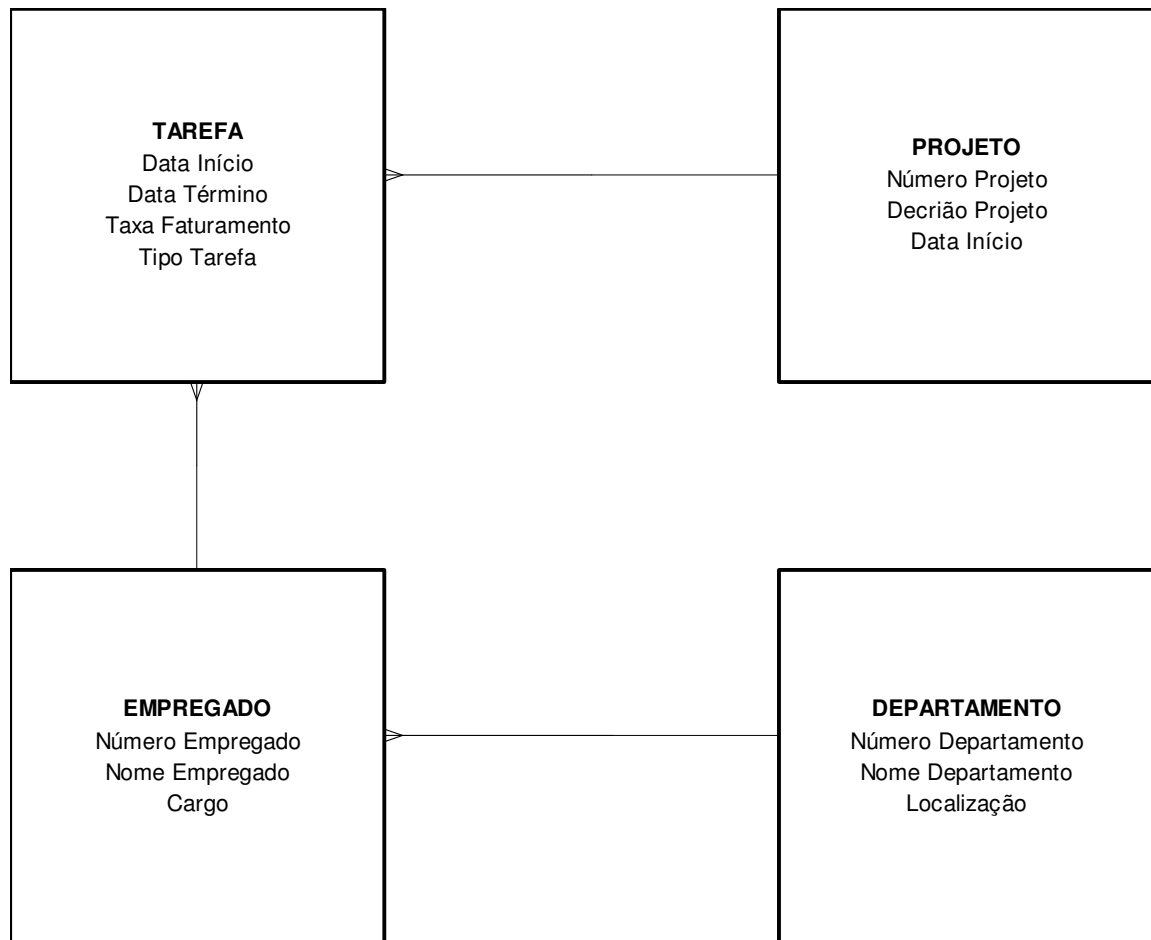
Desenho e Terminologia do Básico Banco de Dados Relacional

O objetivo dessa Unidade é introduzir a você alguns Desenhos e Terminologia do Básico Relacional. Nós não pretendemos ensinar como desenhar um Banco de Dados.

Desenho

Para qualquer Sistema de Banco de Dados que deve realmente ser uma estratégia, analisado, desenhado e construindo por fase. A saída de uma fase começa na entrada da próxima. Nesse caminho erros são encontrados mais facilmente e assim nessa ordem diminuem os custos. A saída do estágio de DESENHO especifica a entrada na fase de CONSTRUÇÃO. Nós temos alcançado o ponto onde precisamos para construir tabelas, como Desenho e Terminologia Básica que precisamos para sermos introduzidos. O Desenho tem o propósito para criar um físico desenho para implementação baseado sobre as saídas das fases analisadas. O Estágio de Análise produz o Diagrama de Entidade e Relacionamento o qual provém uma descrição detalhada do negócio. O modelo e entidade é uma ferramenta de comunicação. Ela é usada para representar o significado da coisa - pessoa, objeto físico, atividades - sobre o qual nós queremos armazenar dados e associar-se entre eles. Essa ordem de modelo é algumas vezes referido como o Modelo Conceitual.

Simples Diagrama de Entidade e Relacionamento



Como ler o Diagrama



Cada departamento pode possuir um ou mais empregado(s).
Cada empregado deve possuir um único departamento.

O Diagrama de Tabelas

O Diagrama de Tabelas provém de descrições da tabela. Basicamente entidade provém um molde para as tabelas do Banco de Dados. O desenho produz inicialmente desenhos das tabelas sobre mapas simples de entidades para as tabelas. Nossa explicação não é para ensinar a você a converter Diagramas para *scripts* de tabelas, mas para mostrar a você como ler um diagrama para ajudar nessa fase. Sobre um Diagrama de Tabelas, tabelas são representadas por caixas. As colunas marcadas e ligadas são escritas sobre a linha. Colunas específicas são escritas nas caixas.

CHAVE PARA UM DIAGRAMA DE TABELA




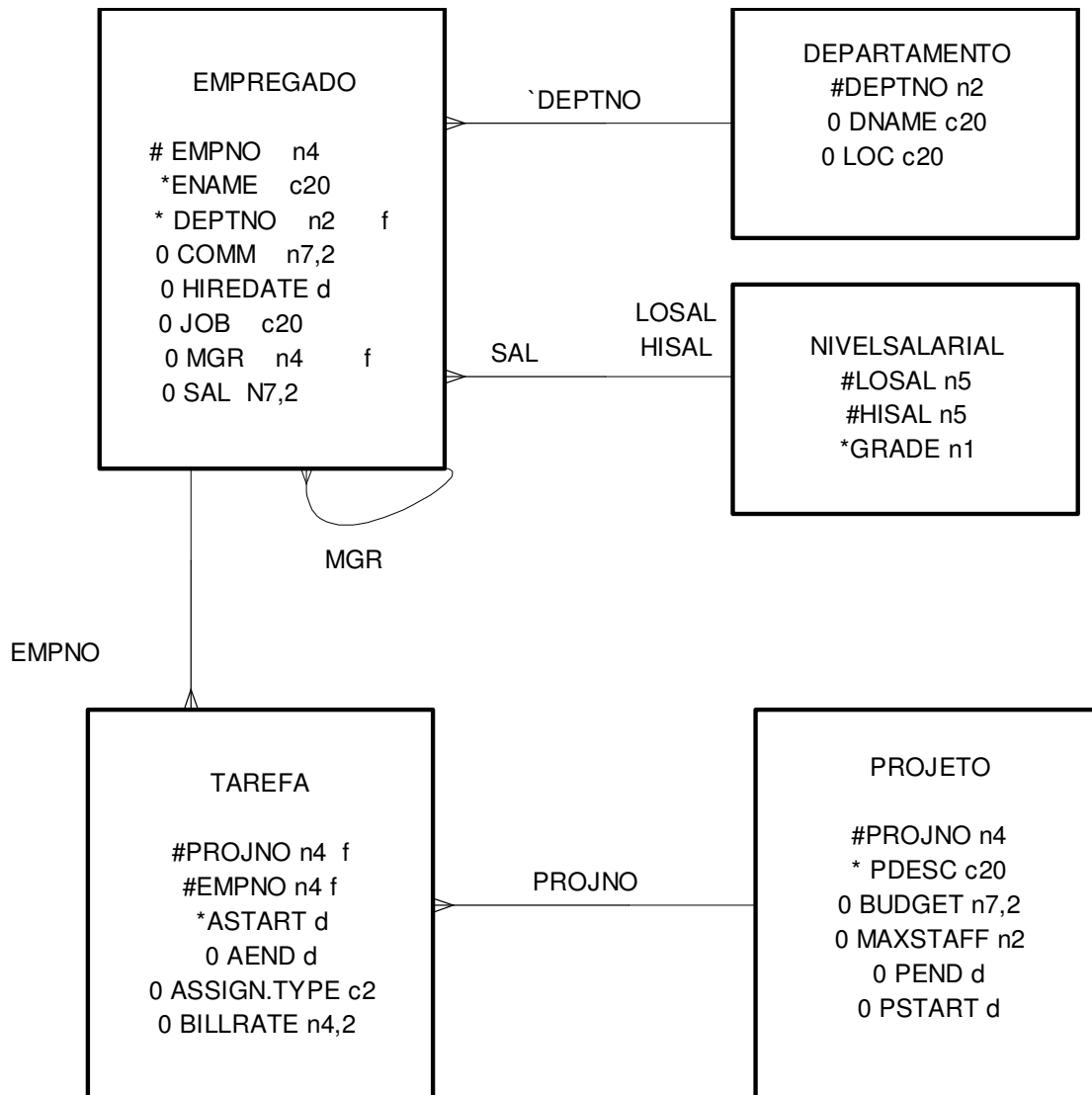
| | |
|-------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| Tabelas | Caixas Retangulares |
| Colunas | Listadas dentro das caixas, únicas identificadas primeiro, obrigatórias e nulas nessa ordem. |
| Formatos das Colunas | c- caracter n-numérica d-date |
| Obrigatória | * não nula assume diferente de nulo |
| coluna chave primária | # |
| coluna opcional | O |
| Coluna chave estrangeira | F |
| Relacionamento | nome da coluna de ligação |
|  | significa um ou mais |
|  | Obrigatório |
|  | Opcional |

Diagrama de Tabelas:

Quando nós estamos desenhando tabelas, devemos especificar uma **chave Primária(Primary Key)**. O # próximo de uma coluna significa que ela faz parte da chave primária. Uma chave primária é uma coluna ou combinação de colunas as quais os valores são únicos e identificam as linhas nas tabelas. Por exemplo, a coluna EMPNO na tabela EMP distingue cada linha. Não deve ter valor duplicado. Se a chave primária for feita de mais de uma coluna, então nenhuma das colunas que fazem parte da chave primária devem ser nulas; então, chave primária deve possuir valor.

Chave Estrangeira(Foreign Key)

O f próximo da coluna indica que ela é uma chave estrangeira. Uma chave estrangeira é uma coluna ou colunas que contém valores de chaves primárias de outras tabelas.

EMPREGADOS

| | | | |
|----------------|------|--------|-----------------------------|
| EMPNO | NAME | ADRESS | Chave estrangeira DEPTNO |
| chave primária | | | |

DEPARTAMENTO

| | |
|----------------|-------|
| DEPTNO | DNAME |
| chave primária | |

Chave Estrangeira permite nós ligarmos tabelas. A coluna DEPTNO na tabela EMP é uma chave estrangeira e tem f próximo dela. Seu valor deve ser igual ao valor da coluna DEPTNO da tabela DEPT a qual tem # próximo e significa que ela é uma coluna chave primária.

Colunas Chaves

Algumas colunas podem ser ambas chaves primárias e estrangeiras.

- DEPTNO é uma chave primária na tabela DEPT, e chave estrangeira na tabela EMP.
- EMPNO é uma chave primária. Ela é então uma chave estrangeira da coluna MGR.

No ORACLE chave primária tem índice único para prevenir a entrada de valores duplicados.

Colunas Obrigatórias e Opcionais

Se uma coluna é prefixada com um *, seu preenchimento é obrigatório.

Uma coluna prefixada com um 0 pode conter valor nulo.

Linguagem de Definição de Dados e Dicionários de Dados

Nessa Unidade, os comandos necessários para criação, remanejamento, adição de comentários e destruição de tabelas são conjuntos de saída. O Dicionário de Dados é então comentado.

Estrutura de Dados ORACLE

- Tabelas podem ser criadas a qualquer momento, igualmente com usuários em tempo real usando o Banco de Dados.
- Tamanho dos dados são variáveis, em que unicamente caracteres/números especificados são armazenados. Espaços e brancos não são armazenados.
- Não existe nenhuma especificação de tamanho para qualquer tabela. Existe definição como o espaço será alocado no Banco de Dados como um todo. Ele é importante, porém, para estimar o espaço ocupado pela tabela usaremos outra vez.
- Estruturas de Tabelas podem ser modificadas em tempo real.

Criando uma Tabela

O nome que você escolhe para uma tabela deve seguir a regra padrão de nomeação de objetos no Banco de Dados ORACLE.

1. O nome deve iniciar com uma letra, A-Z ou a-z.
2. Ele pode conter letras, números, e o caracter especial ' _ ' (underscore). Os Caracteres \$ e # são permitidos mas não são aconselhados.
3. O nome é o mesmo para qualquer forma de escrita maiúscula ou minúscula ou os dois.
4. Ele deve ter no máximo 30 posições.
5. O nome não pode ser igual ao de outra tabela, sinônimo, visão.
6. O nome não deve ser uma palavra reservada do SQL.

| <u>Nome</u> | <u>Válido?</u> |
|--------------|-------------------------------------------------------------------------------------|
| EMP85 | Sim |
| 85EMP | Não, não inicia com uma letra |
| FIXED_ASSETS | Sim |
| FIXED ASSETS | Não, contém um espaço em branco |
| UPDATE | Não, é uma palavra reservada do SQL |
| TABLE1 | Sim, mas você não deve esquecer qual é essa tabela. Escolha um nome mais sugestivo. |

Diretriz para Nomear Tabelas

- Usar nomes sugestivos para tabelas, colunas, índices e outros objetos.

- Ser consistente em abreviações e no uso do singular ou plural na forma de nomear tabelas e colunas,
- Use consistentes regras de nomeação. Uma regra deve ser seguida por todas as tabelas que pertencerem a aplicação FINANCEIRA com FIN_. Consistentes regras de nomeação ajudam os usuários a entender o papel que cada tabela exerce em sua aplicação.
- Use o mesmo nome para descrever a mesma entidade ou atributo por tabela. Por exemplo, a coluna número do departamento das tabelas EMP e DEPT são ambos nomeados como DEPTNO.

Tipos de Colunas

Quando você cria uma tabela, você precisa especificar cada tipo de dado por coluna. A tabela abaixo mostra os mais importantes tipos de dados. Os tipos de dados podem ser acompanhar um ou mais número em parênteses os quais tem informações sobre a largura da coluna. A largura da coluna determina o tamanho máximo que valores na coluna podem ter. Colunas CHAR devem ter o tamanho especificado. Colunas NUMBER podem ter tamanhos especificados, mas não necessariamente.

| <u>Tipos de Dados</u> | <u>Devem Conter</u> |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CHAR(tamanho) | Valores CHAR consiste de letras maiúsculas ou minúsculas, números e caracteres especiais (*,&,%,\$,etc.). Tamanho é a máxima largura, em caracteres de valores dessa coluna, e não pode ser maior que 255. |
| VARCHAR2(tamanho) | CHAR (alfanumérico) dados não maior que 255 caracteres. Tamanho é o máximo número de caracteres que podem ser armazenados em uma coluna VARCHAR2. Na Versão 6 CHAR e VARCHAR2 tem o mesmo efeito. |
| NUMBER | Valor numérico consiste de 1-9 com opcional sinal de + ou - e um ponto decimal. |
| NUMBER(tamanho) | Mesmo que NUMBER mas nesse o tamanho é especificado. Se o tamanho não for especificado o tamanho será assumido como 38. |
| NUMBER(tamanho,decimal) | Mesmos que NUMBER(tamanho), mas especifica o tamanho do decimal. |
| DATE | Valor Data de 14 Janeiro de 712 Antes de Cristo à 14 Dezembro de 4712 depois de cristo. Informações de Horas são armazenadas. |
| LONG | Similar ao CHAR mas o valor deve ser maior que 65535 caracteres. Não mais que uma coluna LONG deve ser especificada em uma tabela. |

Exemplos de Especificações de Colunas

| Especificação | Significado |
|---------------|----------------------------------------------------------------------------------------------------|
| CHAR(12) | coluna deve conter valores char em 12 posições. |
| VARCHAR2(12) | coluna deve conter valores char em 12 posições. |
| NUMER | coluna deve conter valores numéricos |
| NUMER(4) | coluna deve conter valores numéricos com 4 dígitos. |
| NUMER(8,3) | coluna deve conter valores numéricos com 8 dígitos e dígitos a direita a direita do ponto decimal. |
| DATE | coluna deve conter valores de data |
| LONG | coluna deve conter valores longos. |

Criando uma Tabela (CREATE TABLE)

```
CREATE TABLE nome_tabela
(nome_coluna tipo (tamanho) (null/not null),
 nome_coluna tipo (tamanho) (null/not null) ,
...);
```

```
CREATE TABLE DEPT
(DEPTNO NUMBER(2) NOT NULL,
 DNAME CHAR(12),
 LOC CHAR(2));
```

As Opções NULL e NOT NULL

| Opção | Descrição |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| NULL | Essa opção permite valor nulo. Ela é o padrão, e deve ser omitida. |
| NOT NULL | Essa opção força valores na apropriada coluna. Se qualquer coluna NOT NULL for adicionada sem um valor o ORACLE retornará uma mensagem de erro. |

Criando a Tabela EMP

Para criar a tabela EMP, faça:

```
CREATE TABLE EMP
(EMPNO NUMBER(4) NOT NULL,
 ENAME CHAR(10),
 JOB CHAR(10),
 MGR NUMBER(4),
 HIREDATE DATE,
 SAL NUMBER(7,2),
 COMM NUMBER(7,2),
 DEPTNO NUMBER(2) NOT NULL);
```

Table created.

Para ver uma descrição da tabela EMP, faça:

```
DESC EMP;
```

| Name | Null? | Type |
|----------|----------|---------------|
| ----- | ----- | ----- |
| EMPNO | NOT NULL | NUMBER (4) |
| ENAME | | CHAR (10) |
| JOB | | CHAR (10) |
| MGR | | NUMBER (4) |
| HIREDATE | | DATE |
| SAL | | NUMBER (7, 2) |
| COMM | | NUMBER (7, 2) |
| DEPTNO | NOT NULL | NUMBER (2) |

Note os Seguintes pontos:

- EMPNO é definida como NOT NULL, essa coluna unicamente identifica a linha
- DEPTNO é definida como NOT NULL, o valor na coluna é usado para ligar a tabela EMP com a DEPT.
- SAL e COM tem o tipo de dado NUMBER(7,2). Total 7 dígitos. 5 inteiros e dois decimais.

O comando DESCRIBE (DESC) pode ser usado para mostrar a estrutura de qualquer tabela no Dicionário de Dados. Quando você está criando tabelas tipo de colunas podem ser definidas como CHAR ou VARCHAR2. Esse dois tipos de dados tem o mesmo efeito e ambos podem ser usados.

```
CREATE TABLE EMP
(EMPNO NUMBER(4) NOT NULL,
ENAME VARCHAR2(10),
JOB VARCHAR2(10),
MGR NUMBER(4),
HIREDATE DATE,
SAL NUMBER(7,2),
COMM NUMBER(7,2),
DEPTNO NUMBER(2) NOT NULL);
```

Clausula CONSTRAINT

O ORACLE suporta a sintaxe para reservar a integridade sobre as informações sobre checagem de integridade no Dicionário de Dados. Uma reserva de integridade é uma regra que força um relacionamento entre as tabelas do Banco de Dados. Por exemplo, uma reserva de integridade que um empregado na tabela EMP não pode ser atribuído para um departamento que não existe na tabela DEPT. Uma restrição da chave estrangeira (FOREIGN KEY).

CONSTRAINTS podem ser definidas para uma tabela e colunas e são especificadas como parte dos comandos CREATE ou ALTER TABLE.

A propósito das CONSTRAINTS é um conjunto de restrições de valores para validação. Toda a declaração INSERT, UPDATE e DELETE causam uma avaliação da(s) CONSTRAINTS. A CONSTRAINT deve ser satisfeita para ser efetivado o INSERT, UPDATE ou DELETE.

Use CONSTRAINTS para impor uma ou mais das seguintes restrições sobre uma coluna ou um grupo de colunas.

- requerer que uma coluna ou grupo de colunas contenham valores NOT NULL.
- especificar que o valor de uma coluna seja único na tabela referida.
- especificar colunas como CHAVE PRIMÁRIA.
- estabelecer uma restrição de CHAVE ESTRANGEIRA.
- requerer que valor de uma coluna se conforme um valor pré-determinado. (CHECK).

Existem dois tipos de restrições(constraints), de tabelas e colunas. Elas são identificadas igualmente que a restrição de coluna refere-se a um única coluna onde a de tabela pode referenciar-se para uma ou mais colunas. Definições de restrições de tabelas são parte de um global definição de tabela:

```
CREATE TABLE ASSIGNMENT
    (PROJECT NUMBER(4), EMPLOYEE NUMBER(4),
    PRIMARY KEY (PROJECT, EMPLOYEE))
```

Restrições de colunas são locais para a especificação da coluna:

```
CREATE TABLE DEPT (DEPTNO NUMBER PRIMARY KEY ...)
```

Exemplo de Restrições

```
CREATE TABLE EMP
    (EMPNO NUMBER(4) NOT NULL PRIMARY KEY CONSTRAINT EMP_PRIM,
    ENAME VARCHAR2(1) CHECK (ENAME = UPPER(ENAME)),
    JOB VARCHAR2(10),
    MGR NUMBER(4),
    HIREDATE DATE CHECK (HIREDATE <= SYSDATE),
    NI_NUMBER VARCHAR2(12),
    SAL NUMBER(7,2),
    DEPTNO NUMBER(2) NOT NULL FOREIGN KEY (DEPTNO)
    REFERENCES DEPT (DEPTNO)
    CONSTRAINT EMP_DEPT,
    FOREIGN KEY (MGR) REFERENCES EMP (EMPNO) CONSTRAINT EMP_MGR,
    UNIQUE KEY (NI_NUMBER) CONSTRAINT EMP_NI);
```

Notas

1. EMPNO é definido como uma chave primária, e deve então ser definido como NOT NULL

2. A coluna DEPTNO possui uma cláusula REFERENCES que indica que ela é uma chave estrangeira da tabela DEPT. Note que para usar REFERENCES em uma restrição de tabela você precisa ser o criador da tabela ou possuir privilégios sobre a tabela a qual a coluna foi referenciada.

3. CHECK especifica um restrição que deve ser satisfeita. Então a coluna ENAME deve ser digitada em maiúsculo antes de ser inserida na tabela.

4. HIREDATE deve ser maior ou igual a data de hoje para passar a restrição.

5. MGR é identificada como uma chave estrangeira na restrição de tabela e é referenciada a coluna EMPNO.

6. NI_NUMBER é identificada como uma chave única.

Parâmetro das Restrições

| | |
|-----------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CONSTRAINT restrição_nome | especifica o nome da restrição. Este parâmetro é opcional. Se você omiti-lo o nome será atribuído com um nome padrão formado de SYS_Cnome, onde nome é um número que unicamente identifica a restrição. |
| NULL/NOT NULL | especifica que a coluna deve ser ou não preenchida. O padrão é NULL. |
| UNIQUE | especifica que cada linha da tabela deve ter um valor distinto para a coluna. Cada coluna deve ser declarada como NOT NULL e a coluna não deve ser chave primária. |
| PRIMARY KEY | especifica que a coluna é única e identifica cada linha. A coluna deve ser declarada como NOT NULL e não deve ter uma restrição do tipo UNIQUE. A chave primária pode ser composta de várias colunas. Se sua chave primária é uma coluna, você deve usar uma restrição de tabela ou coluna. Se for várias colunas você deve usar uma restrição de tabela. |
| FOREIGN KEY (coluna,...) REFERENCES tabela(col,...)... | identifica as colunas como as chaves estrangeiras das tabelas. Deve referenciar a uma chave primária de uma tabela. Não pode referenciar uma visão. Para referenciar-se a uma tabela você deve ser seu criador ou ter privilégios sobre ela. |
| CHECK | especifica uma condição que uma coluna deve satisfazer para cada linha que existe na tabela. Pode unicamente referenciar-se com colunas da mesma tabela. |

As restrições tem a finalidade de facilitar a integridade de dados na tabela e entre tabelas. A restrição de integridade é forçada para todas as aplicações ou ferramentas que acessem as tabelas. As mais aplicadas restrições são as de NULL e NOT NULL.

Criando uma Tabela com linhas de Outra Tabela.

Existe uma segunda forma de declaração de CREATE TABLE na qual a tabela é criada com linhas derivadas de outra tabela:

```
CREATE TABLE DEPT
[(coluna (NULL/NOT NULL) ...)]
AS SELECT declaração
```

A tabela será criada com as colunas específicas e linhas retiradas sobre a declaração SELECT inseridas nela. Se todas as colunas na declaração SELECT tiverem nome isto é não forem uma expressão, a especificação da coluna pode ser omitida.

Para criar uma tabela DEPT30 com número, nome, cargo e salário do empregado no departamento 30 faça:

```
CREATE TABLE DEPT30
AS
SELECT EMPNO, ENAME, JOB, SAL
FROM EMP
WHERE DEPTNO = 30;
```

Table created.

E para conferir o resultado:

```
DESC DEPT30
```

| Name | Null? | Type |
|-------|----------|--------------|
| ----- | ----- | ----- |
| EMPNO | NOT NULL | NUMBER(4) |
| ENAME | | CHAR(10) |
| JOB | | CHAR(10) |
| SAL | | NUMBER(7, 2) |

Para criar uma tabela contendo nome, salário e faixa do empregado, faça:

```
CREATE TABLE EMP_SALS
(NAME, SALARY, GRADE)
AS
SELECT ENAME, SAL, GRADE
FROM EMP, SALGRADE
WHERE EMP.SAL BETWEEN LOSAL AND HISAL;
```

Table created.

```
DESC EMP_SALS;
```

| Name | Null? | Type |
|--------|-------|--------------|
| ----- | ----- | ----- |
| NAME | | CHAR(10) |
| SALARY | | NUMBER(7, 2) |
| GRADE | | NUMBER |

Para mostra o conteúdo da tabela EMP_SALS, faça:

```
SELECT * FROM EMP_SALS;
```

| NAME | SALARY | GRADE |
|--------|--------|-------|
| ----- | ----- | ----- |
| SMITH | 800 | 1 |
| ADAMS | 1100 | 1 |
| JAMES | 950 | 1 |
| WARD | 1250 | 2 |
| MARTIN | 1250 | 2 |
| MILLER | 1300 | 2 |
| ALLES | 1600 | 3 |
| TURNER | 1500 | 3 |
| JONES | 2975 | 4 |
| BLAKE | 2850 | 4 |
| CLARK | 2450 | 4 |
| SCOTT | 3000 | 4 |
| FORD | 3000 | 4 |
| KING | 5000 | 5 |

Alterando uma Tabela

Use o comando ALTER TABLE para modificar uma definição de tabela.

Use a palavra chave ADD para adicionar uma coluna ou restrição para uma tabela existente.

```
ALTER TABLE nome
ADD (coluna tipo/Restrição)
```

Para adicionar a coluna com o nome da esposa do empregado, faça:

```
ALTER TABLE EMP
ADD (SPOUSES_NAME CHAR(10));
```

Table altered.

Para ver como ficou a tabela faça:

```
DESC EMP
```

| Name | Null? | Type |
|--------------|----------|-------------|
| ----- | ----- | ----- |
| EMPNO | NOT NULL | NUMBER(4) |
| ENAME | | CHAR(10) |
| JOB | | CHAR(10) |
| MGR | | NUMBER(4) |
| HIREDATE | | DATE |
| SAL | | NUMBER(7,2) |
| COMM | | NUMBER(7,2) |
| DEPTNO | NOT NULL | NUMBER(2) |
| SPOUSES_NAME | | CHAR(10) |

Para adicionar uma restrição para uma tabela existente, na qual especifica que o salário mensal não deve exceder 5000 faça:

```
ALTER TABLE EMP
ADD (CHECK (SAL<=5000));
```

Use a palavra chave **MODIFY** para modificar a definição de uma coluna existente.

```
ALTER TABLE nome
MODIFY (coluna tipo (NULL));
```

Para modificar o tamanho de ENAME para 25 caracteres faça:

```
ALTER TABLE EMP
MODIFY (ENAME CHAR(25));
```

Table altered.

DES EMP

| Name | Null? | Type |
|--------------|----------|-------------|
| EMPNO | NOT NULL | NUMBER(4) |
| ENAME | | CHAR(25) |
| JOB | | CHAR(10) |
| MGR | | NUMBER(4) |
| HIREDATE | | DATE |
| SAL | | NUMBER(7,2) |
| COMM | | NUMBER(7,2) |
| DEPTNO | NOT NULL | NUMBER(2) |
| SPOUSES_NAME | | CHAR(10) |

Existem três modificações que você não pode fazer:

1. Você não pode modificar uma coluna que contém nulos para NOT NULL
2. Você não pode adicionar uma coluna que é NOT NULL. Adicione ela nula, preencha todas as linhas e modifique para NOT NULL.
3. Você não pode diminuir o tamanho de uma coluna ou mudar o tipo do dado, a menos que ela não contenha dados.

Para modificar uma restrição você deve excluir ela e então adicionar ela com a modificação.

Use a palavra chave **DROP** para remover uma restrição de uma tabela.

```
ALTER TABLE EMP
DROP CONSTRAINT EMP_NI;
```


Excluindo uma Tabela

Para excluir uma definição de uma tabela no ORACLE usa-se o comando DROP TABLE.

```
DROP TABLE EMP;
```

Excluindo uma tabela perde todos os dados dela e todos os índices associados com ela:

- Todos os dados serão apagados
- Qualquer visão e Sinônimos permanecem, mas eles são inválidos
- Qualquer transação é confirmada
- Somente o criador da tabela ou DBA pode excluí-la.

O Comando COMMENT

Use o comando COMMENT para inserir um comentário não maior de 255 caracteres, sobre uma tabela ou coluna, no Dicionário de Dados.

Para adicionar um comentário na tabela EMP faça:

```
COMMENT ON TABLE EMP IS 'Informações Empregados';
```

Para adicionar um comentário na coluna EMPNO da tabela EMP faça:

```
COMMENT ON COLUMN EMP.EMPNO IS 'Único número do empregado';
```

Para retirar o comentário use o comando COMMENT sem o comentário:

```
COMMENT ON COLUMN EMP.EMPNO IS;
```

Para ver o comentário, selecione a coluna COMMENTS de uma dessas tabelas: ALL_COL_COMMENTS ou USER_COL_COMMENTS. O Dicionário de Dados será comentado depois dessa unidade.

O Comando RENAME

O comando RENAME é usado pelo criador da TABELA, VISÃO e SINÔNIMO para alterar o nome de um objeto do Banco de Dados.

Para renomear um objeto do Banco de Dados a sintaxe é:

```
RENAME velho TO novo
```

Para renomear a tabela EMP para EMPLOYEE, faça:

```
RENAME EMP TO EMPLOYEE
```

É importante notar que qualquer aplicação que se referir para o objeto renomeado terá que ser alterado.

O Dicionário de Dados ORACLE

O Dicionário de Dados é uma das mais importantes partes do ORACLE RDBMS. Ele consiste de um conjunto de tabelas e visões as quais provém da leitura referente ao Banco de Dados. O Dicionário de Dados diz a você:

- o nome dos usuários ORACLE.
- os direitos e privilégios que eles possuem.
- nomes: tabelas, visões, índices, sinônimos, seqüências,...).
- restrições aplicadas para uma tabela.
- informações de auditoria, mostra como quem tem acesso ou pode alterar especificados objetos do Banco de Dados.

O Dicionário de Dados é criado quando o Banco de Dados é criado. Depois confirme o Banco de Dados vai se modificando o Dicionário de Dados vai sendo alterado.

Acessando o Dicionário de Dados

Para acessar os objetos do Dicionário de Dados é via a declaração SQL SELECT pode ser usada para pesquisar informações no Dicionário de Dados.

Nota:

Nenhum usuário pode alterar as linhas ou objeto do Dicionário de Dados pois isso poderia comprometer a integridade do Banco de Dados. O RDBMS já faz todas as alterações necessárias a partir dos comando SQL executados.

Dicionário de Dados Tabelas e Visões

Tabelas

A Base do Dicionário de Dados ou tabelas subjacentes são os primeiros objetos para ser criados no Banco de Dados como eles devem estar presente para todos outros objetos para serem criados. Tabelas do Dicionário de dados são automaticamente criadas pela declaração SQL CREATE DATABASE e são propriedade do usuário SYS. A Base das Tabelas são raramente acessadas diretamente por que as informações não são de fácil entendimento.

Sinônimos

Visões do Dicionário de Dados (Tabelas virtuais). contém informações que possam ser de fácil entendimento para os usuários. Acesso público para o Dicionário de Dados é dado via visões as tabelas.

As três classes de visões são:

| | |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| USER_xxx | objeto propriedade do usuário que pode acessá-la. Por exemplo, visões com esse prefixo seguinte o usuário para mostrar as informações sobre as tabelas criadas pelo usuário e privilégios dados pelo usuário. |
| ALL_xxx | usuário pode usar os objetos os quais tem permissão de acesso, em adicionar objetos os quais lhe pertence. |
| DBA_xxx | para usuários com privilégios de DBA pode acessar qualquer objeto no Banco de Dados. |

Algumas visões de Dicionários de Dados não usam os prefixos acima.

| | |
|--------------------|------------------------------------------------------------------------------------|
| DICTIONARY | lista toas as tabelas do Dicionário de Dados, visões, sinônimos para o usuário. |
| DICT_COLUMNS | mostra todas as colunas que podem ser acessada pelo usuário. |
| CONSTRAINT_DEF | lista todas as restrições entradas para as tabelas que são acessadas pelo usuário. |
| CONSTRAINT_COLUMNS | lista todas as colunas que possuem restrições nomeadas pelo usuário. |

A Visão DICTIONARY

DICTIONARY lista todos os objetos do Dicionário de Dados acessado pelo usuário com uma breve descrição do objeto.

A seguinte declaração SQL mostra os comentários da tabela DICTIONARY:

```
SELECT * FROM DICTIONARY;
```

NOME TABELA

COMENTÁRIO

| | |
|---------------------|----------------------------------------------------------------------|
| ACCESSIBLE_COLUMNS | Colunas de toadas as tabelas, visões e grupo |
| ACCESSIBLE_TABLES | Tabelas e Visões acessíveis pelo usuário |
| ALL_CATALOG | Todas tabelas, visões, sinônimos, seqüências acessíveis pelo usuário |
| ALL_COL_COMMENTS | Comentários sobre colunas acessíveis das tabelas e visões. |
| ALL_COL_GRANTS_MADE | permissões das colunas para qual o usuário é |

| | |
|---------------------|---------------------------------------------------------------------|
| | criador ou proprietário. |
| ALL_COL_GRANTS_RECD | permissões sobre colunas para uso público para os usuários. |
| ALL_CONSTRAINTS | Restrições sobre as tabelas acessíveis |
| ALL_CONS_COLUMNS | restrições sobre as colunas acessíveis |
| ALL_DB_LINKS | Banco de Dados Ligados acessados pelo usuário |
| ALL_DEF_AUDIT_OPTS | Auditoria opção para recentes objetos criados |
| ALL_INDEXES | descrições do índices sobre as tabelas acessadas |
| ALL_IND_COLUMNS | colunas que possuem índice das tabelas. |
| ALL_OBJECTS | Objetos pelo usuário. |
| ALL_SEQUENCES | descrições das seqüências |
| ALL_SYNONYMS | todos os sinônimos |
| ALL_TABLES | descrições de todas as tabelas |
| ALL_TAB_COMMENTS | comentários das tabelas e visões |
| ALL_TAB_GRANTS_MADE | permissões sobre os objetos |
| ALL_TAB_GRANTS_RECD | permissões sobre objetos públicos |
| ALL_USERS | informações sobre os usuários do Banco de Dados |
| ALL_VIEWS | textos de visões |
| AUDIT_ACTIONS | |
| COLUMN_PRIVILEGES | permissões sobre as colunas criadas pelo usuário. |
| CONSTRAINT_COLUMNS | Restrições definidas nas colunas pelas tabelas acessadas |
| CONSTRAINT_DEFS | Restrições definidas nas tabelas acessadas |
| DICTIONARY | descrições das tabelas e visões |
| DICT_COLUMNS | Descrições das colunas das Tabelas e Visões do Dicionário de Dados. |
| DUAL | tabela virtual |
| SYSAUDIT_TRAIL | sinônimo para AUDIT_TRAIL |

| | |
|----------------------|--------------------------------------------------------------------------|
| TABLE_PRIVILEGES | permissões pelos criadores sobre os objetos. |
| USER_AUDIT_CONNECT | auditoria do caminho para o usuário(login) |
| USER_AUDIT_TRAIL | Auditoria dos caminhos relevantes ao usuário |
| USER_CATALOG | Tabelas, Visões, Sinônimos, Seqüências acessíveis pelo usuário. |
| USER_CLUSTERS | descrições das autorizações dos usuários |
| USER_CLU_COLUMNS | Mapa das colunas das tabelas e conjuntos de colunas |
| USER_COL_COMMENTS | Comentários sobre colunas dos tabelas e visões dos usuários |
| USER_COL_GRANTS | Permissões sobre as colunas as quais o usuário é criador. |
| USER_COL_GRANTS_MADE | todas as autorizações sobre as colunas e objetos que o usuário é criador |
| USER_COL_GRANTS_RECD | Autorizações sobre as colunas as quais o usuário é criador. |
| USER_CONSTRAINTS | restrições definidas sobre o acesso das tabelas |
| USER_CONS_COLUMNS | informações sobre acessíveis colunas em definas restrições |
| USER_CROSS_REFS | referencias cruzadas para visões, sinônimos e restrições do usuário |
| USER_DB_LINKS | Banco de Dados ligados pelo usuário |
| USER_EXTENTS | extensões dos segmentos |
| USER_FREE_SPACE | espaço livre na espaço da tabela acessível pelo usuário |
| USER_INDEXES | descrições dos índices criados pelo usuário |
| USER_IND_COLUMNS | colunas que possuem índices |
| USER_OBJECTS | objetos criados pelo usuário |
| USER_SEGMENTS | Armazenamento alocados para todo os segmentos do Banco de Dados. |
| USER_SEQUENCES | Descrições das seqüências do usuário |
| USER_SYNONYMS | as sinônimos usados |
| USER_TABLES | descrições das tabelas do usuários |

| | |
|----------------------|--------------------------------------------------------------------------------|
| USER_TAB_ESPACES | descrição espaço acessado |
| USER_TAB_AUDIT_OPTS | Opções de auditoria para usuários e suas tabelas e visões |
| USER_TAB_COLUMNS | Colunas das tabelas, visões e grupos de usuários |
| USER_TAB_COMMENTS | comentário das tabelas e visões criadas pelo usuário |
| USER_TAB_GRANTS | permissões sobre objetos para quais o usuário é o criador ou possui permissão. |
| USER_TAB_GRANTS_MADE | todas as permissões sobre os objetos criados pelo usuário |
| USER_TAB_GRANTS_RECD | permissões sobre os objetos os quais o usuário é permitido |
| USER_TS_QUOTAS | quota de espaço para o usuário |
| USER_USERS | informações sobre o corrente usuário |
| USER_VIEWS | textos das visões criadas pelo usuário |

Algumas das visões do Dicionário de Dados tem nomes longos. Sinônimos públicos tem o sentido de abreviações, para ajudar o acesso demais comum usuário nas visões do Dicionário de Dados.

As seguintes visões são as mais interessantes para a maioria dos usuários.

| Nome Visão | Sinônimo | Descrição |
|-------------------|----------|-----------------------------------------------------------------------------|
| DICTIONARY | DICT | Lista de todos os objetos do Banco de Dados |
| USER_OBJECTS | OBJ | Objetos criados pelo usuário |
| USER_CATALOG | CAT | Tabelas, visões, sinônimos, seqüências que podem ser acessadas pelo usuário |
| USER_TABLES | TABS | Descrição das tabelas do usuário |
| USER_TAB_COLUMNS | COLS | Colunas das tabelas e visões do usuário |
| USER_COL_COMMENTS | | Comentário das colunas das tabelas e visões do usuário |
| USER_TAB_COMMENTS | | Comentário sobre as tabelas e visões criadas pelo usuário |

| | | |
|-----------------|-----|-----------------------------------------------------------------------------|
| USER_SEQUENCES | SEQ | Descrição das seqüências criadas pelo usuário |
| USER_SYNONYM | SYN | Sinônimos privados do usuário |
| USER_VIEWS | | textos das visões criadas pelo usuário |
| USER_INDEXES | IND | Descrições dos índices usados pelo usuário |
| ALL_OBJECTS | | Objetos que podem ser acessados pelo usuário |
| ALL_TAB_COLUMNS | | Colunas para todas as tabelas e visões que podem ser acessadas pelo usuário |

Para ver a estrutura da tabela USER_OBJECTS faça:

```
DESC USER_OBJECTS
```

| Name | Null? | Type |
|-------------|-------|----------|
| ----- | ----- | ----- |
| OBJECT_NAME | | CHAR(30) |
| OBJECT_ID | | NUMBER |
| OBJECT_TYPE | | CHAR(13) |
| CREATED | | DATE |
| MODIFIED | | DATE |

Uma descrição da Visão DICTIONARY seguinte

```
DESC DICT
```

| Name | Null? | Type |
|------------|-------|-----------|
| ----- | ----- | ----- |
| TABLE_NAME | | CHAR(30) |
| COMMENTS | | CHAR(255) |

Note que o sinônimo abreviado DICT é usado para referir-se a tabela DICTIONARY. Conhecendo a descrição da tabela ou visão você pode construir uma declaração SELECT para mostrar informações sobre um objeto.

```
SELECT OBJECT_NAME, OBJECT_TYPE, CREATED, MODIFIED
FROM USER_OBJECTS
WHERE OBJECT_NAME = 'EMP';
```

| OBJECT_NAME | OBJECT_TYPE | CREATED | MODIFIED |
|-------------|-------------|-----------|-----------|
| ----- | ----- | ----- | ----- |
| EMP | TABLE | 02-MAY-90 | 01-JUL-90 |

Exercício 8 - Linguagem de definição de Dados

Este exercício fala sobre criação de tabelas de Banco de Dados, alteração de definições de Tabelas, e também sobre pesquisas nas tabelas do Dicionário de Dados.

1. Criar a seguinte tabela, você deve abreviar o nome da tabela se requerer, contendo os nomes de colunas especificados.

Nome da Tabela: PROJECTS

| Nome Coluna | Tipo de Dado | Tamanho | Não Nula |
|---------------|--------------|---------|----------|
| PROJID | NUMBER | 4 | NOT NULL |
| P_DESC | CHARACTER | 20 | |
| P_START_DATE | DATE | | |
| P_END_DATE | DATE | | |
| BUDGET_AMOUNT | NUMBER | 7,2 | |
| MAX_NO_STAFF | NUMBER | 2 | |

Nome da Tabela: ASSIGNMENTS

| Nome Coluna | Tipo de Dado | Tamanho | Não Nula |
|--------------|--------------|---------|----------|
| PROJID | NUMBER | 4 | NOT NULL |
| EMPNO | NUMBER | 4 | NOT NULL |
| A_START_DATE | DATE | | |
| A_END_DATE | DATE | | |
| BILL_RATE | NUMBER | 4,2 | |
| ASSIGN_TYPE | CHARACTER | 2 | |

2. Tendo a criação processada com sucesso dessas tabelas, adicionar outra coluna na tabela ASSIGNMENTS:

| Nome Coluna | Tipo de Dado | Tamanho | Não Nula |
|-------------|--------------|---------|----------|
| HOURS | NUMBER | 2 | |

3. Adicionar comentários para as tabelas que você tem criado.

4. Adicionar comentários para coluna PROJID na tabela PROJECTS.

5. Mostrar as colunas especificadas para a tabela USER_TAB_COLUMNS.

7. Mostrar nome tabelas e comentários para qualquer tabela que você tenha criado. Você precisa para pesquisar a USER_COL_COMMENTS ou USER_TAB_COMMENTS tabelas do Dicionário de Dados.

8. Mostrar os nomes das tabelas e comentários para qualquer tabela as quais você tem acesso. Pesquisar a tabela ALL_TAB_COMMENTS.

Respostas dos exercícios propostos

Exercício 1 - Introdução ao SQL, o comando SELECT

- 1- SELECT *
 FROM SALGRADE;
- 2- SELECT *
 FROM EMP;
- 3- SELECT ENAME, DEPTNO, SAL
 FROM EMP
 WHERE SAL BETWEEN 1000 AND 2000;
- 4- SELECT DEPTNO, JOB
 FROM DEPT
 ORDER BY DNAME;
- 5- SELECT DISTINCT JOB
 FROM EMP;
- 6- SELECT *
 FROM EMP
 WHERE DEPTNO IN (10,20)
 ORDER BY ENAME;
- 7- SELECT ENAME, JOB
 FROM EMP
 WHERE JOB = 'CLERK'
 AND DEPTNO = 20;
- 8- SELECT ENAME
 FROM EMP
 WHERE ENAME LIKE '%TH%'
 OR ENAME LIKE '%LL%';
- 9- SELECT ENAME, JOB, SAL
 FROM EMP
 WHERE MGR IS NOT NULL;
- 10- SELECT ENAME, SAL*12+NVL(COMM,0) REMUNERATION
 FROM EMP;
- 11- SELECT ENAME, DEPTNO, HIREDATE
 FROM EMP
 WHERE HIREDATE LIKE '%83';
- 12- SELECT ENAME, SAL*12 ANNUAL_SAL, COMM
 FROM EMP
 WHERE SAL>COMM
 AND JOB = 'SALESMAN'
 ORDER BY SAL DESC, ENAME;
- 13- SELECT ENAME||
 ' HAS HELD THE POSITION OF '||

```

                JOB||
                ' IN DEPT '||
                DEPTNO||
                ' SINCE '||
                HIREDATE "Who, what and when"
FROM            EMP;

14-  SELECT      *
      FROM        EMP
      ORDER BY    NVL(MGR,9999);

```

Exercício 2 - Usando Funções

```

1. SELECT DEPTNO, ENAME, ROUND(SAL*1.15)PCTSAL
   FROM    EMP;

2. SELECT RPAD(ENAME,10)||LPAD(JOB,10) EMPLOYEE_AND_JOB
   FROM    EMP;

3. SELECT ENAME NAME,
          SUBSTR(JOB,1,2)||
          SUBSTR(EMPNO,2,2)||
          SOUNDEX(ENAME CODE
   FROM    EMP;

4. SELECT *
   FROM    EMP
   WHERE   UPPER(JOB) = UPPER('&JOB');

5. SELECT LPAD(' ',(20-LENGTH(DNAME))/2)||DNAME DEPARTMENT
   FROM    DEPT;

6. SELECT ENAME,
          TRANSLATE(SUBSTR(ENAME,1,INSTR(ENAME,'L')), 'L','X')||
          SUBSTR(ENAME,INSTR(ENAME,'L') + 1) FIRST_OCCURRENCE_OF_L
   FROM    EMP;

```

Exercício 3 – Mais Funções

```

1. SELECT ENAME,
          TO_CHAR(HIREDATE,'fmMonth, Ddspth YYYY') date_hired
   FROM    EMP
   WHERE   DEPTNO = 20;

2. SELECT ENAME, HIREDATE, ADD_MONTHS(HIREDATE,12) REVIEW
   FROM    EMP
   ORDER BY ADD_MONTHS(HIREDATE,12);

3. SELECT ENAME,
          DECODE(SIGN(1500-SAL),1,'BELOW 1500',0,'On Target', SAL)
          SALARY
   FROM    EMP
   ORDER BY ENAME;

```

```

4. SELECT TO_CHAR(TO_DATE('&ANYDATE','DD.MM.YY'),'DAY') DAY
   FROM DUAL;

5. DEFINE TIME = MONTHS_BETWEEN(SYSDATE,HIREDATE)
   SELECT ENAME,
          FLOOR(&TIME/12)||' YEARS '||FLOOR(MOD(&TIME,12)|| 'MONTHS '
          "LRNGTH OF SERVICE"
   FROM EMP
   WHERE ENAME = UPPER('&EMPLOYEE_NAME');

6. SELECT '12/34' VALUE,
          DECODE(
            TRANSLATE('12/34','1234567890','9999999999'),
            '99/99','YES','NO') "VALID?:"
   FROM DUAL;

7. SELECT ENAME,
          HIREDATE,
          DECODE(SIGN(TO_CHAR(HIREDATE,'DD')-151,
            NEXT_DAY(LAST_DAY(ADD_MONTHS(HIREDATE,1)), 'FRIDAY')-7,
            NEXT_DAY(LAST_DAY(HIREDATE, 'FRIDAY')-7) PAYDAY
   FROM EMP
   ORDER BY HIREDATE;

ou

SELECT ENAME, HIREDATE,
       NEXT_DAY(LAST_DAY(ROUND(HIREDATE,'MONTH')-7, 'FRIDAY')) PAYDAY
   FROM EMP
   ORDER BY HIREDATE;

```

Exercício 4 - Funções de Grupo

```

1. SELECT MIN(SAL) MINIMUM
   FROM EMP;

2. SELECT MAX(SAL), MIN(SAL), AVG(SAL)
   FROM EMP;

3. SELECT JOB,
          MAX(SAL) MAXIMUM,
          MIN(SAL) MINIMUM,
   FROM EMP
   GROUP BY JOB;

4. SELECT COUNT(1) MANAGERS
   FROM EMP
   WHERE JOB = 'MANAGER';

5. SELECT JOB,
          AVG(SAL) AVSAL,
          AVG(SAL*12+NVL(COMM,0)) AVCOMP
   FROM EMP
   GROUP BY JOB;

```

```
6. SELECT    MAX(SAL) - MIN(SAL) DIFFERENCE
   FROM      EMP;

7. SELECT    DEPTNO, COUNT(1)
   FROM      EMP
   GROUP BY  DEPTNO
   HAVING    COUNT(1) > 3;

8. SELECT    EMPNO
   FROM      EMP
   GROUP BY  EMPNO
   HAVING    COUNT(1) > 1;

9. SELECT    MGR, MIN(SAL)
   FROM      EMP
   GROUP BY  MGR
   HAVING    MIN(SAL) >= 1000
   ORDER BY  MIN(SAL);
```

Exercício 5 - Simples Ligações (Join)

```
1. SELECT ENAME, DNAME
   FROM   EMP, DEPT
   WHERE  EMP.DEPTNO = DEPT.DEPTNO;

2. SELECT ENAME, E.DEPTNO, DNAME
   FROM   EMP E, DEPT D
   WHERE  E.DEPTNO = D.DEPTNO;

3. SELECT ENAME, LOC LOCATION, DNAME
   FROM   EMP, DEPT
   WHERE  EMP.DEPTNO = DEPT.DEPTNO
   AND    SAL > 1500;

4. SELECT ENAME, JOB, SAL, GRADE
   FROM   EMP, SALGRADE
   WHERE  SAL BETWEEN LOSAL AND HISAL;

5. SELECT ENAME, JOB, SAL, GRADE
   FROM   EMP, SALGRADE
   WHERE  SAL BETWEEN LOSAL AND HISAL
   AND    GRADE = 3;

6. SELECT ENAME, SAL, LOC LOCATION
   FROM   EMP, DEPT
   WHERE  EMP.DEPTNO = DEPT.DEPTNO
   AND    LOC = 'DALLAS';

7. SELECT ENAME, JOB, SAL, GRADE, DNAME
   FROM   EMP, SALGRADE, DEPT
   WHERE  EMP.DEPTNO = DEPT.DEPTNO
   AND    SAL BETWEEN LOSAL AND HISAL
   AND    JOB <> 'CLERK'
   ORDER BY SAL DESC;
```

Exercício 6 - Outros Métodos de Ligações

1.

```
SELECT EMPNO, ENAME, DNAME, LOC
FROM   EMP E, DEPT D
WHERE  E.DEPTNO = D.DEPTNO;
```
2.

```
SELECT D.DEPTNO, DNAME
FROM   EMP E, DEPT D
WHERE  E.DEPTNO(+) = D.DEPTNO
AND    E.EMPNO IS NULL;
```
3.

```
SELECT EMPS.EMPNO,
       EMPS.ENAME,
       MGRS.EMPNO MGRNO,
       MGRS.ENAME MGR_NAMR
FROM   EMP EMPS, EMP MGRS
WHERE  EMPS.MGR = MGRS.EMP;
```
4.

```
SELECT EMPS.EMPNO,
       EMPS.ENAME,
       MGRS.EMPNO MGRNO,
       MGRS.ENAME MGR_NAMR
FROM   EMP EMPS, EMP MGRS
WHERE  EMPS.MGR = MGRS.EMP (+);
```
5.

```
SELECT JOB
FROM   EMP
WHERE  HIREDATE BETWEEN '01-JAN-83' AND '30-JUN-83'
INTERSECT
SELECT JOB
FROM   EMP
WHERE  HIREDATE BETWEEN '01-JAN-84' AND '30-JUN-84';
```
6.

```
SELECT E.ENAME EMPLOYEE,
       E.HIREDATE,
       M.ENAME MANAGER,
       M.HIREDATE
FROM   EMP E, EMP M
WHERE  E.MGR = M.EMPNO
AND    E.HIREDATE < M.HIREDATE;
```
7.

```
SELECT DEPTNO, DNAME
FROM   DEPT
MINUS
SELECT EMP.DEPTNO, DNAME
FROM   EMP, DEPT
WHERE  EMP.DEPTNO = DEPT.DEPTNO;
```
8.

```
SELECT ENAME,
       JOB,
       SAL*12+NVL(COMM,0) ANNUAL_SAL,
       D.DEPTNO,
       DNAME,
       GRADE
FROM   EMP E, SALGRADE, DEPT D
WHERE  E.DEPTNO = D.DEPTNO
```

```

AND      SAL BETWEEN LOSAL AND HISAL
AND      (SAL*12+NVL(COMM,0) = 36000
OR       E.JOB = 'CLERK')
ORDER BY E.JOB;

```

Exercício 7 - Sub-pesquisas

1.

```

SELECT JOB, ENAME, SAL
FROM   EMP
WHERE  (SAL,JOB) IN
        (SELECT MAX(SAL), JOB
         FROM EMP
         GROUP BY JOB)
ORDER BY SAL DESC;

```
2.

```

SELECT ENAME, JOB, SAL
FROM   EMP
WHERE  (SAL,JOB) IN
        (SELECT MIN(SAL), JOB
         FROM EMP
         GROUP BY JOB)
ORDER BY SAL;

```
3.

```

SELECT DEPTNO, ENAME, HIREDATE
FROM   EMP
WHERE  (HIREDATE,DEPTNO) IN
        (SELECT MAX(HIREDATE), DEPTNO
         FROM EMP
         GROUP BY DEPTNO)
ORDER BY HIREDATE;

```
4.

```

SELECT ENAME, SAL SALARY, DEPTNO
FROM   EMP E
WHERE  SAL >
        (SELECT AVG(SAL)
         FROM EMP
         WHERE DEPTNO = E.DEPTNO)
ORDER BY DEPTNO;

```
5.

```

SELECT DEPTNO, DNAME
FROM   DEPT
WHERE  DEPTNO NOT IN
        (SELECT DEPTNO
         FROM EMP);

```
6.

```

DEFINE REM = SAL*12+NVL(COMM,0)
SELECT DEPTNO, SUM(&REM) COMPENSATION
FROM   EMP
GROUP BY DEPTNO
HAVING SUM(&REM) =
        (SELECT MAX(SUM(&REM))
         FROM EMP
         GROUP BY DEPTNO);

```
7.

```

SELECT ENAME, SAL

```

-
- ```

FROM EMP E
WHERE 3 >
 (SELECT COUNT(1)
 FROM EMP
 WHERE E.SAL < SAL);

```
8. COLUMN YEAR FORMAT A4  
COLUMN NUMBER\_OF\_EMPS FORMAT 9 HEADING 'NUMBER OF EMPS'
- ```

SELECT TO_CHAR(HIREDATE, 'YYYY' YEAR,
              COUNT(EMPNO) NUMBER_OF_EMPS
FROM EMP
GROUP BY TO_CHAR(HIREDATE, 'YYYY')
HAVING COUNT(EMPNO) =
      (SELECT MAX(COUNT(EMPNO))
       FROM EMP
       GROUP BY TO_CHAR(HIREDATE, 'YYYY'));

```
9. COLUMN SALARY FORMAT 999,999.99
COLUMN DEPT_AVG LIKE SALARY
BREAK ON DEPTNO ON DEPT_AVG
- ```

SELECT E.ENAME ENAME, E.SAL SALARY, E.DEPTNO, AVG(A.SAL) DEPT_AVG
FROM EMP A, EMP E
WHERE E.DEPTNO = A.DEPTNO
AND E.SAL >
 (SELECT AVG(SAL)
 FROM EMP
 WHERE DEPTNO = E.DEPTNO)
GROUP BY E.ENAME, E.SAL, E.DEPTNO
ORDER BY AVG(A.SAL);

```
10. SELECT ENAME, HIREDATE, '\*' MAXDATE  
FROM EMP  
WHERE HIREDSTE = (SELECT MAX(HIREDATE) FROM EMP)  
UNION  
SELECT ENAME, HIREDATE, ' '  
FROM EMP  
WHERE HIREDATE <> (SELECT MAX(HIREDATE) FROM EMP);

### **Exercício 8 - Linguagem de definição de Dados**

- ```

1. CREATE TABLE PROJECTS
    (PROJID          NUMBER(4) NOT NULL,
     P_DES           CHAR(20),
     P_START_DATE    DATE,
     P_END_DATE      DATE,
     BUFGET_AMOUNT   NUMBER(7,2),
     MAX_NO_SATFF    NUMBER(2));

CREATE TABLE ASSIGNMENTS
    (PROJID          NUMBER(4) NOT NULL,
     EMPNO           NUMBER(4) NOT NULL,
     A_START_DATE    DATE,
     A_END_DATE      DATE,

```

```

        BILL_RATE          NUMBER(4,2),
        ASSIGN_TYPE        CHAR(2));

2.    ALTER TABLE ASSIGNMENTS
      ADD (HOURS NUMBER(2));

3.    COMMENT ON TABLE ASSIGNMENTS IS 'Unique Project details';

COMMENT ON TABLE ASSIGNMENTS IS 'Assignments for any employee
on a project':

4.    COMMENT ON COLUMN PROJECTS.PROJID IS 'Unique identifier for
a project':

5.    DESCRIBE USER_TAB_COLUMNS

Name                           Null?                  Type
-----
TABLE_NAME                     NOT NULL              CHAR(30)
COLUMN_NAME                    NOT NULL              CHAR(30)
DATA_TYPE                      NOT NULL              CHAR(9)
DATA_LENGTH                    NOT NULL              NUMBER
DATA_PRECISION                 NOT NULL              NUMBER
DATA_SCALE                     NOT NULL              NUMBER
NULLABLE                       NOT NULL              CHAR(1)
COLUMN_ID                      NOT NULL              NUMBER
DEFAULT_LENGTH                 NOT NULL              NUMBER
DATA_DEFAULT                   NOT NULL              LONG

6.    SELECT A.TABLE_NAME, A.COLUMN_NAME, B.COMMENTS
      FROM USER_TAB_COLUMNS A, USER_TAB_COMMENTS B
      WHERE A.TABLE_NAME = B.TABLE_NAME;
```