

Unidade 6: Introdução ao SQL

1. Nesta Aula

Alterando tabelas existente
Inserir, alterar e excluir registros
Consultas de seleção de registros utilizando o comando SELECT
Criar pontos para salvar ou desfazer alterações
Testar as regras de validações e relacionamentos aprendidos
Apresentar os operadores BETWEEN, IN, LIKE e IS NULL

Metas de Compreensão

Conhecer como altera as tabelas existentes. Aprender os comandos para criar pontos de referência e desfazer alterações. Conhecer como você pode inserir, alterar, excluir e recuperar dados no SGBD.

Apresentação

O objetivo nesta aula é aprender a alterar tabelas que já existem no banco de dados. Você deve entender como inserir, alterar e excluir registros utilizando o SQL. Conhecer os comandos COMMIT e ROLLBACK vai lhe ajudar desfazer algumas operações que podem ocorrer. Também você iniciará o comando SELECT que retorna dados que foram armazenados em tabelas. Por fim, você vai ver como testar as restrições que você criou e aprender a utilizar os operadores do SQL.

Para entender estes conceitos, esta unidade está organizada da seguinte forma:

- a seção 2 apresenta como alterar tabelas existente num banco de dados;
- a seção 3 mostra o comando INSERT e apresentar os conceitos da SELECT
- a seção 4 apresenta e expressões aritméticas e os operadores BETWEEN, IN, LIKE e IS NULL;
- a seção 5 apresenta UPDATE e DELETE;

Não deixe de utilizar as bibliografias associadas à unidade. Bom estudo!

2. Alterando Tabelas Existente

Nesta aula você vai aprender como alterar estruturas de tabelas mudando características de atributos e adicionando colunas. Todas as alterações na estrutura de uma tabela são feitas utilizando o comando **ALTER TABLE**, seguido por uma palavra que informa o tipo de alteração que se deseja. Existem três opções diferentes para alterar. A opção **ADD** adiciona uma coluna ou restrição, **MODIFY** altera as características de uma coluna e **DROP** exclui uma coluna ou restrição da tabela. Se a tabela estiver vazia todas as operações serão executadas sem problemas, mas se a tabela tiver registro, o SGBD vai verificar se a ação é possível de ocorrer sem perda de dados, se não for uma mensagem negando a operação será exibida. Assim operações como DROP em uma coluna com registros será sempre negada. A sintaxe básica para alterar uma tabela inserindo é:

```
ALTER TABLE nome_da_tabela  
ADD [nome_da_coluna tipo_de_dados] | [restrição]
```

Para exemplificar imagine que você tenha criado a tabela TB_PROVA como o seguinte código:

```
CREATE TABLE TB_PROVA (  
id_prova          NUMBER(5) ,  
dt_prova          DATE ,  
st_prova          VARCHAR2(20) ,  
CONSTRAINT tb_prova_id_prova_pk PRIMARY KEY (id_prova)  
);
```

Em algum momento durante a análise você perceberá que será melhor ter o atributo hr_prova separado de dt_prova para gerar relatórios. Para alterar a tabela e acrescentar uma nova coluna utilize:

```
ALTER TABLE TB_PROVA
ADD hr_prova DATE;
```

Observe que para modificar um campo que não estava como obrigatório para obrigatório basta utilizar o código:

```
ALTER TABLE TB_PROVA
MODIFY dt_prova NOT NULL;
```

Se você quiser modificar o tipo de um atributo, utilize:

```
ALTER TABLE TB_PROVA
MODIFY hr_prova TIMESTAMP;
```

A modificação do tamanho de um tipo de dados se dá com o seguinte código:

```
ALTER TABLE TB_PROVA
MODIFY st_prova VARCHAR2(10);
```

Note que você pode aumentar o tamanho mesmo existindo registro na tabela, contudo a redução só é possível se todos os registros não utilizam o tamanho especificado. No exemplo o campo st_prova tinha a possibilidade de ter até 20 caracteres, a redução para 10 só será efetivada após o SGBD verificar se nenhum dado será perdido com a solicitação. Uma utilização muito comum é utilizar o comando ALTER TABLE para adicionar as restrições ao se criar uma tabela. Isso se dá para facilitar encontrar erros ao criar as tabelas. Para ilustrar imagine que você agora vai acrescentar as tabelas apresentadas na figura 01 na base de dados.

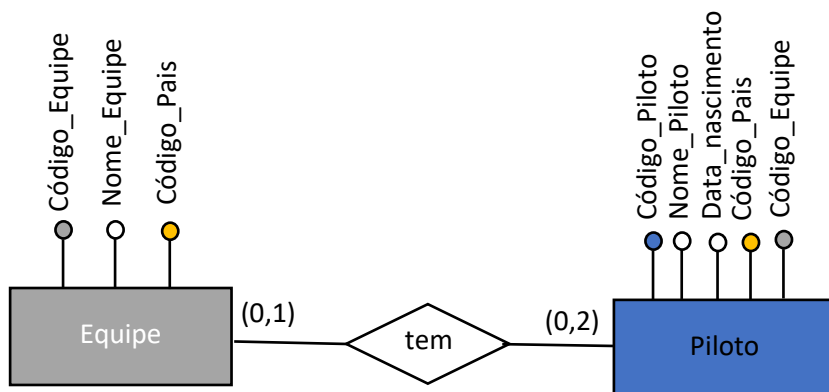


Figura 01. DER F1 acrescentando TB_EQUIPE

Assim poderíamos criar a TB_EQUIPE como exemplificado a seguir.

```
CREATE TABLE TB_EQUIPE (
  id_equipe   NUMBER(5),
  nm_equipe   VARCHAR2(50),
  id_pais     NUMBER(5)
);
```

Observe que no exemplo você não informou qual é a chave primária, qual é a chave estrangeira, se terá campos obrigatórios ou qualquer outra restrição. Você vai utilizar o comando ALTER TABLE para definir cada uma dessas restrições. Assim se você errar ao declarar uma delas, será mais fácil encontrar o erro e corrigi-lo. Segue como fica o restante do código:

```
ALTER TABLE TB_EQUIPE
ADD CONSTRAINT tb_equipe_id_equipe_pk
PRIMARY KEY (id_equipe);
```

```
ALTER TABLE TB_EQUIPE
ADD CONSTRAINT tb_equipe_id_pais_fk
FOREIGN KEY (id_pais)
REFERENCES TB_PAIS (id_pais);
```

```
ALTER TABLE TB_EQUIPE
ADD CONSTRAINT tb_equipe_nm_equipe_nn
CHECK (nm_equipe IS NOT NULL);
```

Na aula anterior você já tinha criado a TB_PILOTO com o código:

```
CREATE TABLE TB_PILOTO (
id_piloto      NUMBER(5),
nm_piloto      VARCHAR2(50),
dt_nascimento  DATE DEFAULT SYSDATE,
id_pais        NUMBER(5),
ie_sexo        CHAR(1),
CONSTRAINT tb_piloto_ie_sexo_ck  CHECK (ie_sexo IN ('M','F')),
CONSTRAINT tb_piloto_id_piloto_pk PRIMARY KEY (id_piloto),
CONSTRAINT tb_piloto_id_pais_fk  FOREIGN KEY (id_pais)
REFERENCES TB_PAIS (id_pais),
CONSTRAINT tb_piloto_nm_piloto_ck CHECK (nm_piloto = UPPER(nm_piloto))
);
```

Observe que neste caso ficou faltando uma coluna id_equipe, assim altere com o código:

```
ALTER TABLE TB_PILOTO
ADD id_equipe NUMBER(5);

ALTER TABLE TB_PILOTO
ADD CONSTRAINT tb_piloto_id_equipe_fk
FOREIGN KEY (id_equipe)
REFERENCES TB_EQUIPE (id_equipe);
```

Também é possível utilizar o comando ALTER TABLE para remover uma coluna ou restrição. A sintaxe seria a seguinte:

```
ALTER TABLE nome_da_tabela
DROP [PRIMARY KEY|COLUMN nome_da_coluna|CONSTRAINT nome_restricao];
```

Observe que para remover uma restrição você deve informar o nome da restrição. Por isso, uma boa prática é sempre criar restrições com nome e não deixar que o SGBD crie de forma automática. Se acontecer de você deixar o SGBDR Oracle criar e precisar consultar o nome atribuído, utilize o comando SELECT apresentado a seguir.

```
SELECT * FROM USER_CONSTRAINTS
WHERE table_name='TB_PROVA';
```

Onde aparece o TB_EQUIPE você poderá substituir pelo nome de qualquer tabela que desejar consultar. Outro comando semelhante é:

```
SELECT constraint_name, constraint_type
FROM USER_CONSTRAINTS
WHERE table_name='TB_PROVA';
```

Algumas informações sobre as restrições que foram criadas para uma tabela são armazenadas na tabela de controle e atualizada automaticamente. Esta tabela é a USER_CONSTRAINTS. O comando SELECT faz uma busca dos dados cadastrados e exibindo o resultado. Agora que você sabe o nome de suas restrições você pode experimentar apagar uma coluna com o comando:

```
ALTER TABLE TB_PROVA  
  DROP COLUMN hr_prova;
```

O comando vai apagar a coluna hr_prova da tabela. Para validar a alteração, utilize o comando:

```
DESC TB_PROVA;
```

Para apagar uma restrição utilize:

```
ALTER TABLE TB_PROVA  
  DROP CONSTRAINT SYS_C007125;
```

Observe que como não definimos o nome da CONSTRAINT quando solicitamos que o SGBD alterasse a TB_PROVA tornando o campo dt_prova obrigatório o SGBD atribuiu o nome de forma automática. No seu sistema pode ter criado outro nome, assim verifique com a instrução SELECT anterior. Como você percebeu não é bom deixar nomes estranhos, assim utilize o comando para corrigir:

```
ALTER TABLE TB_PROVA  
  ADD CONSTRAINT tb_prova_dt_prova_nn  
  CHECK (dt_prova IS NOT NULL);
```

Para finalizar o comando ALTER TABLE vale ressaltar que em alguns momentos você pode precisar desabilitar algumas restrições criadas no banco de dados. São momentos em que você vai ter uma grande carga de processamento, normalmente de balanço ou exportação de dados para uma base de estatística, onde não há necessidade de normalização. Se você tiver certeza que não ocorrerá modificação na base de dados, você pode acelerar o processo desabilitando algumas restrições ou todas. O comando a seguir desabilita uma restrição:

```
ALTER TABLE nome_da_tabela  
  DISABLE nome_da_restricao;
```

Por exemplo para desabilitar a chave estrangeira da TB_EQUIPE digite:

```
ALTER TABLE TB_EQUIPE  
  DISABLE CONSTRAINT tb_prova_id_pais_fk;
```

Para habilitar a chave estrangeira digite:

```
ALTER TABLE TB_EQUIPE  
  ENABLE CONSTRAINT tb_prova_id_pais_fk;
```

Se você desativar uma restrição de chave única ou primária que esteja usando um índice exclusivo, a Oracle descartará o índice exclusivo. Quando você habilitar a chave primária o SGBD vai criar o índice automaticamente. Através do comando ALTER TABLE você consegue mudar a tabela do modo gravação para consulta.

```
ALTER TABLE TB_EQUIPE READ ONLY;
```

Este comando não muda a tabela para somente leitura, impedindo que alterações com comandos DML e DDL ocorram. Para voltar a tabela para o modo gravação utilize o comando apresentado a seguir.

```
ALTER TABLE TB_EQUIPE READ WRITE;
```

A tabela volta ao modo gravação, permitindo que se insira, altere, apague sua estrutura e dados.

3. INSERT e conceitos iniciais da SELECT

Para inserir uma linha em uma tabela SQL exige a utilização do comando INSERT. A sintaxe básica do **INSERT** é:

```
INSERT INTO nome_da_tabela [(nome_da_coluna [,nome_da_coluna])]  
VALUES (valor_da_coluna [,valor_da_coluna]);
```

Para exemplificar veja como você pode inserir um país na TB_PAIS.

```
INSERT INTO TB_PAIS VALUES (1,'Brasil','207.7 mi');
```

No exemplo foi informado o nome_da_tabela como TB_PAIS. Observe que não foi atribuído as informações de quais colunas e em qual ordem seriam enviados os dados no VALUES. Assim, o SGBD assume que serão enviados todos os dados na ordem que foram informados quando se criou a tabela. A instrução **VALUES** passa a informar os valores, sendo a ordem id_pais, nm_pais, nr_populacao. O id_pais é a chave primária, isso quer dizer que ele não pode ser nulo ou ser um valor repetido na tabela. Ele é do tipo NUMBER o que dispensa o uso de aspas simples. Todos os dados do tipo VARCHAR2 devem ser informados com as aspas simples e não podem ultrapassar o limite ou tamanho que foi definido na estrutura da tabela. Veja uma variação que você pode utilizar.

```
INSERT INTO TB_PAIS VALUES (2,'Alemanha',NULL);
```

Neste caso foi escolhido que seriam enviados todos os dados na ordem em que se criou a tabela, mas no VALUES o usuário optou por deixar o valor do atributo nr_populacao nulo utilizando a instrução **NULL** para esse fim. Outra forma de inserir seria:

```
INSERT INTO TB_PAIS (id_pais, nm_pais) VALUES (3,'Finlândia');
```

Você pode mudar a ordem em que os dados serão informados, como segue no próximo comando.

```
INSERT INTO TB_PAIS (nm_pais, id_pais) VALUES ('França',4);
```

Pode conter todos os parâmetros em uma outra ordem.

```
INSERT INTO TB_PAIS (nr_populacao, id_pais, nm_pais)  
VALUES ('5.731 mi',5,'Dinamarca');
```

Pode também informar a ordem e deixar um valor nulo.

```
INSERT INTO TB_PAIS (nr_populacao, id_pais, nm_pais)  
VALUES (NULL,6,'Bélgica');
```

Observe que o valor nulo só será aceito se no momento em que você definiu as CONSTRAINT você não criou uma restrição de campo obrigatório ou NOT NULL. Para inserir múltiplas linhas utilize o comando INSERT ALL.

```
INSERT ALL  
INTO TB_PAIS (id_pais, nm_pais) VALUES (7,'Espanha')  
INTO TB_PAIS (id_pais, nm_pais) VALUES (8,'Reino Unido')  
INTO TB_PAIS (id_pais, nm_pais) VALUES (9,'Austrália')  
SELECT * FROM DUAL;
```

O comando **INSERT ALL** é usado para adicionar várias linhas com uma única instrução INSERT. As linhas podem ser inseridas em uma tabela ou em várias tabelas usando apenas um comando SQL. No exemplo inserimos três linhas ou tuplas na TB_PAIS. Observe que no final do comando INSERT ALL existe uma SELECT da tabela DUAL. Esta tabela apresenta uma única coluna chamada DUMMY e um único registro. Ela foi criada no usuário SYS e sua função é auxiliar em operações como a do comando INSERT ALL. Mais adiante você verá outras operações que a tabela DUAL pode auxiliar. Outra forma de utilizar o INSERT ALL é utilizar a ordem padrão da tabela.

INSERT ALL

```
INTO TB_PAIS VALUES (10, 'Países Baixos', NULL)
INTO TB_PAIS VALUES (11, 'Suécia', NULL)
INTO TB_PAIS VALUES (12, 'Canadá', NULL)
INTO TB_PAIS VALUES (13, 'Itália', NULL)
INTO TB_PAIS VALUES (14, 'Rússia', NULL)
SELECT * FROM DUAL;
```

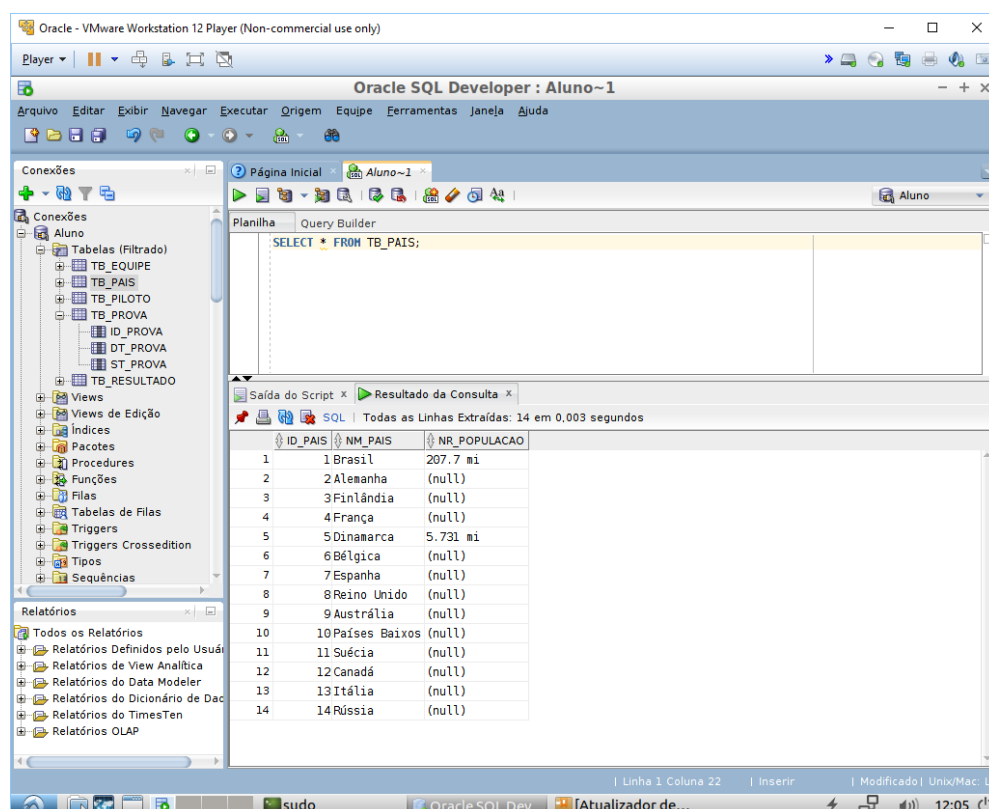
Para verificar como os dados, ou o conteúdo que está na tabela você deve utilizar o comando SELECT. Este é principal comando da SQL e será necessárias diversas interações com o bando de dados para absorver todas as possibilidades e combinações implementadas nesta instrução. Para iniciar observe uma descrição simplificada de sua sintaxe.

```
SELECT lista_de_colunas FROM nome_da_tabela;
```

Esta é uma sintaxe bem simplificada que introduz a SELECT onde a lista_de_colunas representa um ou mais atributos que devem ser exibidos. Se você optar por solicitar mais de uma coluna, você deve separar as colunas utilizando uma vírgula. Também é permitido utilizar o caractere asterisco como coringa, ou seja, todas as colunas da tabela na ordem de criação participaram da consulta. Por exemplo se você deseja ver todos os países que cadastrou na TB_PAIS pode utilizar:

```
SELECT * FROM TB_PAIS;
```

O resultado é apresentado na figura 02 do comando SELECT * FROM TB_PAIS.



Oracle - VMware Workstation 12 Player (Non-commercial use only)

Oracle SQL Developer : Aluno~1

Arquivo Editar Exibir Navegar Executar Origem Equipe Ferramentas Janela Ajuda

Conexões

Aluno

Tabelas (Filtrado)

- TB_EQUIPE
- TB_PAIS
- TB_PILOTO
- TB_PROVA
- ID_PROVA
- DT_PROVA
- ST_PROVA
- TB_RESULTADO

Views

Views de Edição

Índices

Pacotes

Procedures

Funções

Filas

Tabelas de Filas

Triggers

Triggers Crossedition

Tipos

Sequências

Relatórios

Todos os Relatórios

- Relatórios Definidos pelo Usuário
- Relatórios de View Analítica
- Relatórios de Data Modeler
- Relatórios do Dicionário de Dados
- Relatórios do TimesTen
- Relatórios OLAP

Planilha Query Builder

```
SELECT * FROM TB_PAIS;
```

Salida do Script x Resultado da Consulta x

SQL | Todas as Linhas Extraídas: 14 em 0,003 segundos

ID_PAIS	NM_PAIS	NR_POPULACAO
1	Brasil	207.7 mi
2	Alemanha	(null)
3	Finlândia	(null)
4	França	(null)
5	Dinamarca	5.731 mi
6	Bélgica	(null)
7	Espanha	(null)
8	Reino Unido	(null)
9	Austrália	(null)
10	Países Baixos	(null)
11	Suécia	(null)
12	Canadá	(null)
13	Itália	(null)
14	Rússia	(null)

Linha 1 Coluna 22 | Inserir | Modificado | Unix/Mac | LF

sudo Oracle SQL Dev... [Atualizador de... 12:05

Figura 02. Resultado da SELECT * FROM TB_PAIS

Embora o comando SELECT possa ser agrupado em uma única linha, em outras consultas as sequencias vão ficar mais complexas e por isso serão apresentadas em linhas separadas. Utilizar uma convenção para escrever e ler SQL torna mais fácil a manutenção, rastreamento lógico, correções e o reuso de código SQL.

Ao executar uma consulta a uma tabela o SGBD retorna um conjunto de linhas que apresentam as mesmas características que uma tabela. A SELECT é orientada para atuar sobre o conjunto de linhas podendo atuar a uma ou mais colunas e zero ou mais linhas de uma tabela. Para solicitar apenas os nomes dos países cadastrados na TB_PAIS escreva:

```
SELECT nm_pais FROM TB_PAIS;
```

Quando mostramos o resultado de uma pesquisa, normalmente é retornado o nome das colunas selecionadas como cabeçalho. Observe na figura 03 como é apresentado o nome do campo NM_PAIS.

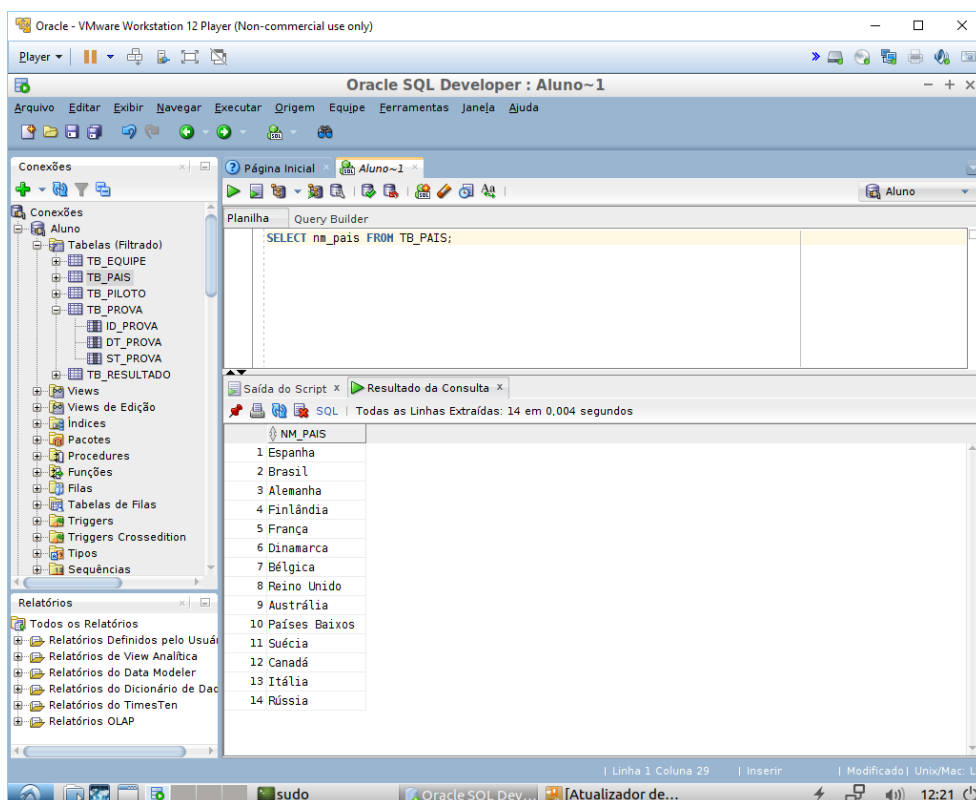


Figura 03. Nome do campo NM_PAIS

Muitas vezes não é aceitável exibir o nome do atributo ou o retorno como título da coluna. Para alterar o nome da coluna é possível criar um apelido para a coluna ou ALIAS. O ALIAS irá substituir o nome da coluna, protegendo a estrutura do SGBD e facilitando a exibição em relatórios gerados pelos usuários. A sintaxe do ALIAS é bem simples, você poderá optar por utilizar o comando **AS** ou não. Se você quiser substituir o nome da coluna por apenas uma palavra simples basta digitar após o nome da coluna, mas, caso queira utilizar uma sentença ou caracteres especiais deverá utilizar aspas-duplas para indicar que o nome deve ser substituído pela sentença. Veja a sintaxe do comando ALIAS:

```
SELECT nome_da_coluna [AS] apelido FROM nome_da_tabela;
```

Onde:

AS é opcional e serve para indicar que após virá o apelido da coluna; apelido pode ser uma única palavra e neste caso não será necessário utilizar aspas-duplas ou poderá ser uma sentença entre aspas-duplas.

Para exemplificar veja todas as variações permitidas


```

SELECT nm_pais nome FROM TB_PAIS;
SELECT nm_pais AS nome FROM TB_PAIS;
SELECT nm_pais "Nome" FROM TB_PAIS;
SELECT nm_pais AS "Nome" FROM TB_PAIS;
SELECT nm_pais "Nome do País" FROM TB_PAIS;
SELECT nm_pais AS "Nome do País" FROM TB_PAIS;

```

Todas as variações apresentadas acima são aceitáveis e você deve estar preparado para utilizar a definida como padrão para o SGBD. Além do uso do ALIAS em uma SELECT é possível utilizar os operadores aritméticos em um atributo, quer seja numérico ou data. Para exemplificar você vai inserir na TB_PILOTO o piloto Nico Hülkenberg. Para inserir o atributo dt_nascimento será necessário utilizar a instrução **TO_DATE** que converte uma sequência de caracteres de texto para o formato data. Digite a instrução:

```
INSERT INTO TB_EQUIPE VALUES (1, 'Renault', 4);
```

```

INSERT INTO TB_PILOTO VALUES (
    1, --id_piloto chave primária
    'NICO HÜLKENBERG', --nm_piloto deve escrever em maiúsculas
    TO_DATE('19/08/1987', 'dd/mm/yyyy'), --dt_nascimento
    3, --id_pais pais em que nasceu
    'M', --ie_sexo pode ser M ou F
    1 --id_equipe chave estrangeira
);

```

Observe a única instrução nova é o TO_DATE que é responsável por converter a string para o formato date. A sintaxe do TO_DATE é:

```
TO_DATE( string [,mascara]);
```

Onde:

string é o texto que você quer converter para o formato DATE

mascara é um campo opcional no qual você informa o que o sistema deve encontrar na string

A máscara pode apresentar uma combinação de valores como apresentado na tabela 01.

Tabela 01. Elementos que podem ser utilizados na mascara

Parâmetro	Explicação
YEAR	Enunciado do Ano
YYYY	Ano de 4 dígitos
YYY YY Y	3, 2 ou último dígito (s) do ano.
IYY IY I	3, 2 ou último dígito (s) do ano no padrão ISO.
IYYY	Ano de 4 dígitos com base no padrão ISO
RRRR	Aceita um ano de 2 dígitos e retorna um ano de 4 dígitos. Um valor entre 0-49 retornará um ano 20xx. Um valor entre 50-99 retornará um ano 19xx.
Q	Trimestre do ano (1, 2, 3, 4; JAN-MAR = 1)
MM	Meses de 01-12 sendo JAN = 01
MON	Nome abreviado do mês
MONTH	Nome do mês, preenchido com espaços em branco com um comprimento de 9 caracteres.
RM	Numeral Romano (I-XII; JAN = I)
WW	Semana do ano (1-53), onde a primeira semana começa no primeiro dia do ano e continua até o sétimo dia do ano.

Tabela 01. continuação

Parâmetro	Explicação
W	Semana do mês (1-5), onde a primeira semana começa no primeiro dia do mês e termina no sétimo.
IW	Semana do ano (1-52 ou 1-53) com base no padrão ISO.
D	Dia da semana (1-7).
DAY	Nome do dia.
DD	Dia do mês (1-31).
DDD	Dia do ano (1-366).
DY	Nome abreviado do dia.
J	Dia juliano, o número de dias desde 01 de janeiro de 4712 AC.
HH	Hora do dia (1-12).
HH12	Hora do dia (1-12).
HH24	Hora do dia (0-23).
MI	Minute (0-59).
SS	Segundo (0-59).
SSSSS	Segundos após a meia-noite (0-86399).
AM, A.M., PM ou P.M.	Indicador de meridiano
AD ou A.D.	Indicador Ano Domini ou Era Comum ou E.C.
BC ou B.C.	Indicador de Before Christ, ou seja, Antes de Cristo ou A.C.
TZD	Informações sobre a economia diurna. Por exemplo, 'PST'
TZH	Horário do fuso horário.
TZM	Minuto do fuso horário.
TZR	Região do fuso horário.

Após ter inserido o piloto você pode experimentar fazer cálculos com operadores aritméticos na SELECT. É possível utilizar operadores aritméticos nos atributos ou em uma expressão condicional. Expressões Aritméticas podem conter nome de colunas, valores numéricos constantes e operadores aritméticos. Os operadores são apresentados na tabela 02.

Tabela 02. Operadores Aritméticos

Operadores	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão

É importante não confundir o símbolo de multiplicação com o coringa. Algumas implementações do SQL utilizam o caractere asterisco para ambas as funções e consegue diferenciar seu uso pela lógica dos parâmetros passados. Ao utilizar os operadores aritméticos todas as regras de precedências são válidas e caso queira modificar a ordem utilize os parênteses. Apenas para lembrar se você não modificar a ordem as operações serão realizadas como apresentado na tabela 03.

Tabela 03. Ordem de execução das operações aritméticas

Ordem	Operações
1º	Entre parênteses
2º	Potencialização
3º	Multiplicações e divisões
4º	Somas e subtrações

Para experimentar teste o seguinte comando:

```
SELECT dt_nascimento+7 AS "Uma semana após nascer" FROM TB_PILOTO;
```

Você pode experimentar utilizar outra operação, por exemplo, para atual idade do piloto.

```
SELECT (sysdate-dt_nascimento)/365.2425 AS "Idade" FROM TB_PILOTO;
```

Neste exemplo o SGBD irá pegar a data atual do servidor e subtrair a data de nascimento apresentado o resultado em dias. Após a operação que estava entre parênteses o SGBD irá calcular a idade em anos, por realizar a divisão dos dias vividos por 365.2425 para obter a idade em anos. Além das operações aritméticas é possível concatenar colunas com outras colunas ou com strings utilizando ||. A sintaxe é:

```
SELECT coluna||[coluna/string] AS alias FROM nome_da_tabela;
```

Para exemplificar, experimente:

```
SELECT nm_piloto || ' nasceu em ' || dt_nascimento
      AS "Nascimento dos pilotos da F1"
FROM TB_PILOTO;
```

4. Expressões aritméticas e os operadores BETWEEN, IN, LIKE e IS NULL

Agora que você tem uma noção básica do funcionamento da SELECT você pode conhecer a sintaxe da instrução. O comando SELECT apresenta a seguinte sintaxe:

```
SELECT [distinct] col1 [alias], coln
FROM tab1 [alias], tabn [alias]
WHERE condição
GROUP BY colunas
HAVING condição
ORDER BY expressão ou chave [desc];
```

Não se preocupe que você vai estudar e ver cada um dos comandos apresentados na sintaxe do SELECT. A Cláusula WHERE indica qual condição de execução o SELECT deve retornar. Ela funciona com os operadores apresentados na tabela 04.

Tabela 04. Operadores para clausula WHERE

Operador	Descrição
=	Igual
<>	Diferente
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual

Para exemplificar você pode solicitar que seja exibido todos os países que tenha o nome Brasil cadastrado na tabela.

```
SELECT nm_pais AS "Nome do País", id_pais AS "Código do País"
FROM TB_PAIS
WHERE nm_pais = 'Brasil';
```

Observe que nas condições os dados alfanuméricos e datas presentes na cláusula WHERE devem estar entre aspas simples. Também é importante salientar que no mundo real, a busca de dados normalmente vai envolver diversas condições. Por exemplo você pode querer saber quais pilotos nasceram em um determinado país e tem mais que 25 anos. Para isso será necessário utilizar operadores lógicos. Os operadores **AND** e **OR** devem ser usados para fazer composições de expressões lógicas. O predicado AND esperará que ambas as condições sejam verdadeiras enquanto o predicado OR esperará uma das condições seja verdadeira. 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 posteriormente todos os ORs serão feitos. O SQL tem três operadores lógicos que são apresentados na tabela 05.

Tabela 05. Operadores Lógicos

Operadores	Descrição
AND	E
OR	OU
NOT	NÃO

Caso queira garantir uma determinada condição seja realizada utilize os parênteses. Os parênteses especificam, a ordem na qual os operadores devem ser avaliados (prioridade). Sempre que você estiver em dúvida sobre qual dos dois operadores será feito primeiro quando a expressão é avaliada, use sempre parênteses para definir a prioridade das expressões.

Operadores Especiais

No SQL existem cinco operadores que operam, com todos tipos de dados. A tabela 06 apresenta esses operadores.

Tabela 06. Operadores especiais para todos os tipos de dados

Operador	Significado
BETWEEN [vl_inicial] AND [vl_final]	Entre dois valores (inclusive)
IN (lista)	Comparar uma lista de valores
LIKE	Compara um parâmetro alfanumérico
IS NULL	É um valor nulo
EXISTS	Utilizado para verificar se uma subconsulta retorna alguma linha

O operador **BETWEEN** pode ser utilizado para verificar se um valor está dentro de uma faixa de valores. Por exemplo se você quiser listar todos os países que tenham a chave primária entre 2 e 8 utilize:

```
SELECT id_pais, nm_pais
FROM TB_PAIS
WHERE id_pais BETWEEN 2 AND 8;
```

Por via de regra o menor valor na clausula BETWEEN deve ser sempre listado primeiro e o maior valor deve ser listado após o AND. Outro operador **IS NULL** verifica quais campos em uma tabela estão com valores nulos. Assim a SELECT vai retornar apenas as linhas ou tuplas com o atributo nulo. Por exemplo se quiser solicitar a lista de países que não apresentam número da população.

```
SELECT id_pais, nm_pais
FROM TB_PAIS
WHERE nr_populacao IS NULL;
```

O operador especial **LIKE** é utilizado como coringa para encontrar padrões ou strings em um atributo. Algumas vezes você precisa procurar valores que você não conhece exatamente. Usando o operador LIKE é possível selecionar linhas combinando parâmetros alfanuméricos. O caractere de porcentagem ou % é utilizado como coringa nas pesquisas de strings. Também é possível utilizar o underscore ou simplesmente _ para criar o coringa de um caractere. Experimente o comando:

```
SELECT nm_pais
FROM TB_PAIS
WHERE nm_pais LIKE 'A%';
```

No exemplo todos os países que começam com a letra A em seu nome, serão listados. Você pode pedir que se procure uma parte da string, como apresentado a seguir:

```
SELECT nm_pais
FROM TB_PAIS
WHERE nm_pais LIKE 'Br%';
```

Ou pode solicitar que tenha o final de acordo com a regra da consulta. O comando a seguir utiliza o coringa na frente da string fazendo que o final seja observado e esteja de acordo com a condição.

```
SELECT nm_pais
FROM TB_PAIS
WHERE nm_pais LIKE '%ia';
```

Outra opção é colocar o coringa antes e depois de uma sequência de caracteres fazendo que apenas as tuplas que tiverem parte da string no meio dela sejam retornadas. O comando a seguir apresenta um caso semelhante de uso.

```
SELECT nm_pais
FROM TB_PAIS
WHERE nm_pais LIKE '%ema%';
```

A utilização do underscore tem a mesma sintaxe do percentual contudo apenas um caractere será substituído. O exemplo seguinte irá retornar todos as linhas que podem satisfazer a condição _rasil.

```
SELECT nm_pais
FROM TB_PAIS
WHERE nm_pais LIKE '_rasil';
```

Lembre-se que você pode utilizar combinações livremente, apresentando em um único LIKE _ e % simultaneamente.

```
SELECT nm_pais
FROM TB_PAIS
WHERE nm_pais LIKE '%a_ca';
```

Lembre-se que a maioria dos SGBDs implementa o SQL como não sensitive-case mas diferenciam os caracteres dos dados armazenados. Isso quer dizer que BRASIL, Brasil e brasil tem caracteres ASCII diferentes e retornem valores diferentes. Para evitar problemas você pode criar uma regra de restrição que pode exigir uma determinada forma ao inserir os dados. Em outra aula você verá uma outra forma de solucionar este problema, transformando momentaneamente a cadeia de caracteres em maiúsculas ou minúsculas. O operador especial IN cria uma lista de valores. Seu uso é semelhante ao operador OR, sendo que todos os valores contidos na lista do operador **IN** devem ser do mesmo tipo de dados. Cada um dos valores da lista será testado e comparado ao atributo, se o valor corresponder a qualquer valor da lista a tupla é selecionada.

```
SELECT nm_pais, id_pais
FROM TB_PAIS
WHERE id_pais IN (1,4,8,10);
```

Outra forma de realizar a mesma seleção

```
SELECT nm_pais, id_pais
FROM TB_PAIS
WHERE nm_pais IN ('Brasil','Países Baixos','Reino Unido','França');
```

O operador especial IN receberá maior atenção quando você for aprender exclusivamente sobre as subconsultas. Por fim, o operador especial **EXISTS** é útil para criar condições onde se deseja saber se existe dados em uma outra consulta. Ou seja, se a subconsulta retornar qualquer tupla então a consulta principal será executada.

```
SELECT nm_piloto, id_pais
FROM TB_PILOTO
WHERE EXISTS (
```

```

SELECT *
FROM TB_PAIS
WHERE id_pais=3
);

```

A subconsulta será executada primeira e como ela retorna uma tupla ou linha, a consulta principal pode ser executada. Como o operador IN o operador EXISTS é utilizado com subconsultas que serão vistos em outra aula.

Retornando a sintaxe do SELECT você notará que falta ver a cláusula **ORDER BY**. Ela serve para ordenar uma lista de forma ordenada. Normalmente a ordem das linhas retornadas de uma pesquisa é indefinida. A cláusula ORDER BY pode ser usada para ordenar as linhas. Se você for usar o comando ORDER BY lembre-se que ele deve ser sempre a última instrução da cláusula de declaração de uma SELECT. Por padrão a ordem da instrução ORDER BY é sempre crescente, mas você pode solicitar a ordem decrescente utilizando DESC. Não é necessário definir a ordem crescente, mas se você quiser a instrução ASC declara de forma explícita. Para testar a instrução digite:

```

SELECT nm_pais, id_pais
FROM TB_PAIS
ORDER BY nm_pais;

```

Outra forma seria:

```

SELECT nm_pais, id_pais
FROM TB_PAIS
ORDER BY nm_pais ASC;

```

Se quiser a ordem decrescente utilize:

```

SELECT nm_pais, id_pais
FROM TB_PAIS
ORDER BY nm_pais DESC;

```

Para avaliar a ordenação de múltiplas colunas você deve inserir mais alguns pilotos. Assim digite:

```

INSERT INTO TB_PILOTO VALUES (2,'JOLYON PALMER',
                                TO_DATE('20/01/1991', 'dd/mm/yyyy'),
                                8,'M',1
);

INSERT INTO TB_EQUIPE VALUES (2,'Ferrari',13);

INSERT INTO TB_PILOTO VALUES (3,'SEBASTIAN VETTEL',
                                TO_DATE('03/07/1987', 'dd/mm/yyyy'),
                                2,'M',2
);

```

Agora com três pilotos você pode avaliar ordenação de múltiplas colunas. É possível utilizar mais de uma coluna na cláusula ORDER BY. O limite de colunas é o número de colunas da tabela. Na cláusula ORDER BY especifica-se as colunas que serão ordenadas, separando as por vírgula. Se algumas ou todas serão invertidas especifique DESC depois de cada uma das colunas. Para ordenar por duas colunas digite o com o comando:

```

SELECT id_equipe, nm_piloto
FROM TB_PILOTO
ORDER BY id_equipe, nm_piloto;

```

Neste caso o id_equipe será o primeiro campo ordenado e quando ocorrer repetição de dados a segunda coluna será ordenada. Como no exemplo existem dois pilotos na mesma equipe

será ordenado primeiro as equipes e depois os nomes dos pilotos. Outra forma de escrever o mesmo comando é substituir o nome das colunas que devem ser ordenadas pelo número ou ordem em que aparecem no retorno de dados. Veja a instrução equivalente a anterior:

```
SELECT id_equipe, nm_piloto
FROM TB_PILOTO
ORDER BY 1, 2;
```

A figura 04 apresenta o resultado da SELECT. Esta notação é bem útil quando a definição das colunas é muito extensa ou apresenta formulas e cálculos complexos.

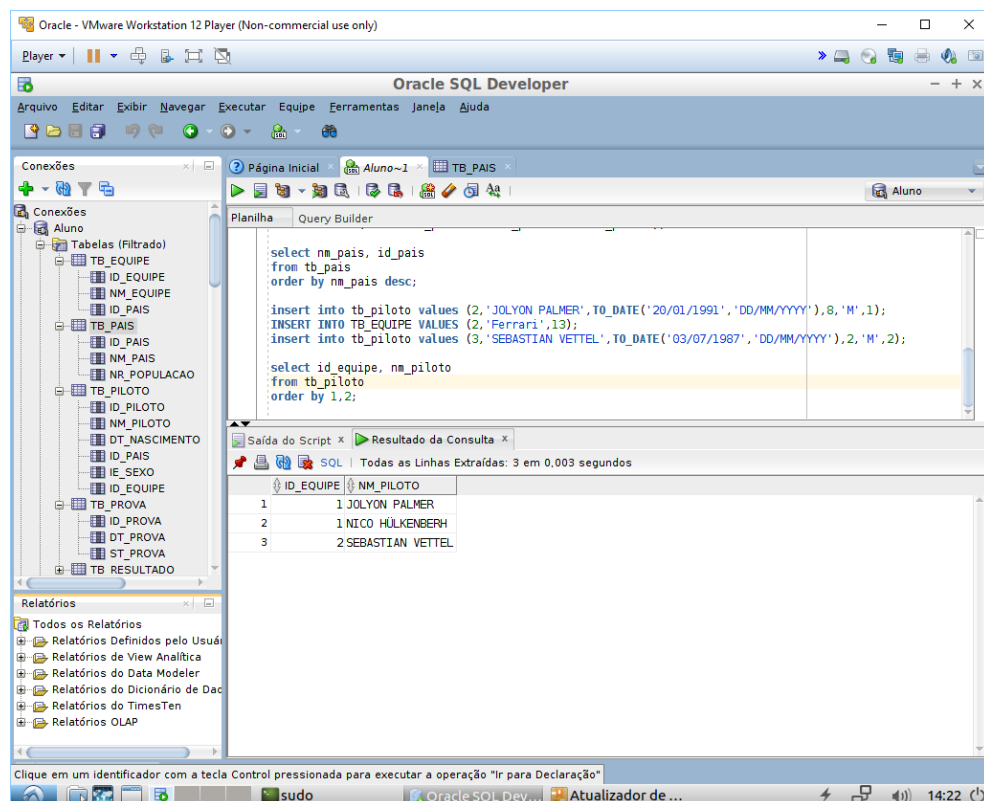


Figura 04. Resultado da SELECT com ORDER BY

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 SGBD. Lembre-se que esse comando não altera a ordem dos dados que estão armazenados no banco de dados.

Além da ordenação o comando SELECT pode solicitar dados distintos ou diferentes. A cláusula **DISTINCT** produz uma lista de valores distintos, ou seja, sem repetições. Por exemplo:

```
SELECT id_equipe
FROM TB_PILOTO;
```

Retorna os dados apresentados na tabela 07:

Tabela 07. Todos os códigos de equipe na tabela TB_PILOTO

ID_EQUIPE
1
1
2

Observe se utilizar o comando:

```
SELECT DISTINCT id_equipe
FROM TB_PILOTO;
```

O retorna os dados é diferente, pois serão filtrados as linhas duplicadas, conforme apresentado na tabela 08:

Tabela 08. Código de equipe distintos da tabela TB_PILOTO

ID_EQUIPE
1
2

Finalizando a sessão no SQL todas as expressões podem ser negativas com o uso do comando NOT. Com ele você pode criar expressões como NOT BETWEEN para indicar tudo que estiver fora da faixa, NOT IN para indicar tudo que não estiver na lista, NOT LIKE para indicar tudo que não conter a linha de caracteres e IS NOT NULL indicando tudo que não for nulo.

5. UPDATE e DELETE

Para atualizar, alterar ou modificar os dados em uma tabela você irá utilizar o comando **UPDATE**. A sintaxe desse comando é:

```
UPDATE nome_da_tabela
SET nome_da_coluna = novo_valor [, nome_da_coluna = novo_valor]
[WHERE lista_de_condição];
```

Por exemplo, você pode inserir os dados de alguns pilotos com o código SQL abaixo:

```
INSERT INTO TB_EQUIPE VALUES (3, 'McLaren-Honda', 8);
```

```
INSERT INTO TB_EQUIPE VALUES (4, 'Williams-Mercedes', 8);
```

```
INSERT INTO TB_PILOTO VALUES (4, 'KIMI RÄIKKÖNEN',
    TO_DATE('17/10/1979', 'dd/mm/yyyy'),
    2, 'M', 2
);
```

```
INSERT INTO TB_PILOTO VALUES (5, 'STOFFEL VANDOORNE',
    TO_DATE('03/07/1987', 'dd/mm/yyyy'),
    2, 'M', 3
);
```

```
INSERT INTO TB_PILOTO VALUES (6, 'FERNANDO ALONSO',
    TO_DATE('29/07/1981', 'dd/mm/yyyy'),
    7, 'M', 3
);
```

Após a inserção dos dados notar que o país de origem está errado. Assim para corrigir:

```
UPDATE TB_PILOTO
SET id_pais = 3
WHERE id_piloto = 4;
```

E se precisar corrigir mais de uma tributo pode utilizar:

```
UPDATE TB_PILOTO
SET dt_nascimento = TO_DATE('26/03/1992', 'dd/mm/yyyy'),
    id_pais = 6
WHERE id_piloto = 5;
```

Observe que é importante definir muito bem a clausula WHERE, pois a ausência desta clausula provocaria alterações em todos os dados armazenados na tabela TB_PILOTO. Portanto tome cuidado e sempre especifique a clausula WHERE quando for utilizar o comando UPDATE para

não alterar sem querer todas as linhas da tabela especificada. Sempre confirme as correções utilizando o comando SELECT. Para verificar se o comando UPDATE não afetou todas as linhas da tabela TB_PILOTO, digite:

```
SELECT * FROM TB_PILOTO;
```

É fácil excluir uma linha ou tupla da tabela. Para apagar os dados em uma tabela você utiliza o comando DELETE que apresenta a seguinte sintaxe:

```
DELETE FROM nome_da_tabela  
[WHERE lista_de_condição];
```

Para exemplificar você pode inserir dados na TB_PROVA para apagar em seguida. Utilize o comando:

```
INSERT INTO TB_PROVA VALUES  
(1,TO_DATE('12/11/2017','dd/mm/yyyy'),'cancelado');  
INSERT INTO TB_PROVA VALUES  
(2,TO_DATE('26/03/2017','dd/mm/yyyy'),'realizado');
```

Agora que existe duas linhas você poderá apagar uma com o comando:

```
DELETE FROM TB_PROVA  
WHERE id_prova = 2;
```

O comando **DELETE** irá apagar a prova de 26 de março de 2017. É importante você identificar corretamente qual tupla deseja apagar, pois se mais de uma tupla satisfizer a condição, está também será apagada. Por isso no exemplo foi utilizado a chave primária, para identificar de forma inequívoca, ou seja, sem erros a tupla desejada. Caso você queira apagar todas as linhas de uma tabela, mas manter sua estrutura, digite:

```
DELETE FROM TB_PROVA;
```

Este comando deve ser utilizado com cuidado, pois apaga todas as linhas da tabela. Antes de utilizar o comando DELETE, confirme a operação visualizando as linhas que serão atualizadas com o comando SELECT, pois ambos apresentam a mesma condição WHERE. Nunca omita a cláusula WHERE. No caso da omissão, todos os registros da tabela serão eliminados.

O comando **CREATE TABLE AS SELECT** cria uma tabela com sua estrutura e dados tendo como baseado em um comando SELECT.

```
CREATE TABLE TB_PAIS  
AS SELECT * FROM TB_TEMP;
```

Outro comando que você pode precisar é o **RENAME**. Este comando é utilizado nos casos de alteração de nome das tabelas. Sua sintaxe é bem simples:

```
RENAME nome_antigo_tabela TO nome_novo_tabela;
```

Para exemplificar

```
RENAME TB_TEMP TO TB_PAIS_BK;
```

Outro comando interessante presente na linguagem SQL é a capacidade que dispomos de cancelar ou gravar uma série de ações. Depois que você inicia uma sessão o SGBD cria um ponto de referência para executar uma sequência de ações no SQL. Uma transação é um conjunto de operações, delimitadas por um início e um fim. Iniciando quando se executa o primeiro comando SQL e terminando de acordo com a situação. Os comandos COMMIT e ROLLBACK são responsáveis por nos ajudar nesta tarefa. O comando **COMMIT** quando executado grava todas as alterações no banco de dados ou os efeitos dos comandos de uma

transação como INSERT, DELETE E UPDATE. A sintaxe é simples tanto para o comando COMMIT:

COMMIT;

Quando você cria uma sessão todas as suas alterações só estão visíveis para você enquanto estiver utilizando a interface. Se você quer tornar perene suas alterações ou visíveis para outros usuários você deve utilizar o comando COMMIT. É interessante notar que quando você sai de uma sessão e a mesma é encerrada sem problemas, ocorre um COMMIT implícito. Se sua sessão finaliza por qualquer outro problema, ocorre um ROLLBACK implícito. Além disso, todo comando DDL tal como CREATE, ALTER e DROP e DCL como GRANT e REVOKE provocam o fim da transação corrente, havendo um COMMIT implícito.

Se o comando COMMIT não foi utilizado para armazenar de forma perene as alterações é possível restaurar o banco de dados até seu último ponto de COMMIT ou ROLLBACK. O comando **ROLLBACK** desfaz quaisquer alterações desde o último COMMIT e retorna os dados aos valores existentes antes das alterações. A sintaxe para o comando ROLLBACK é bem simples:

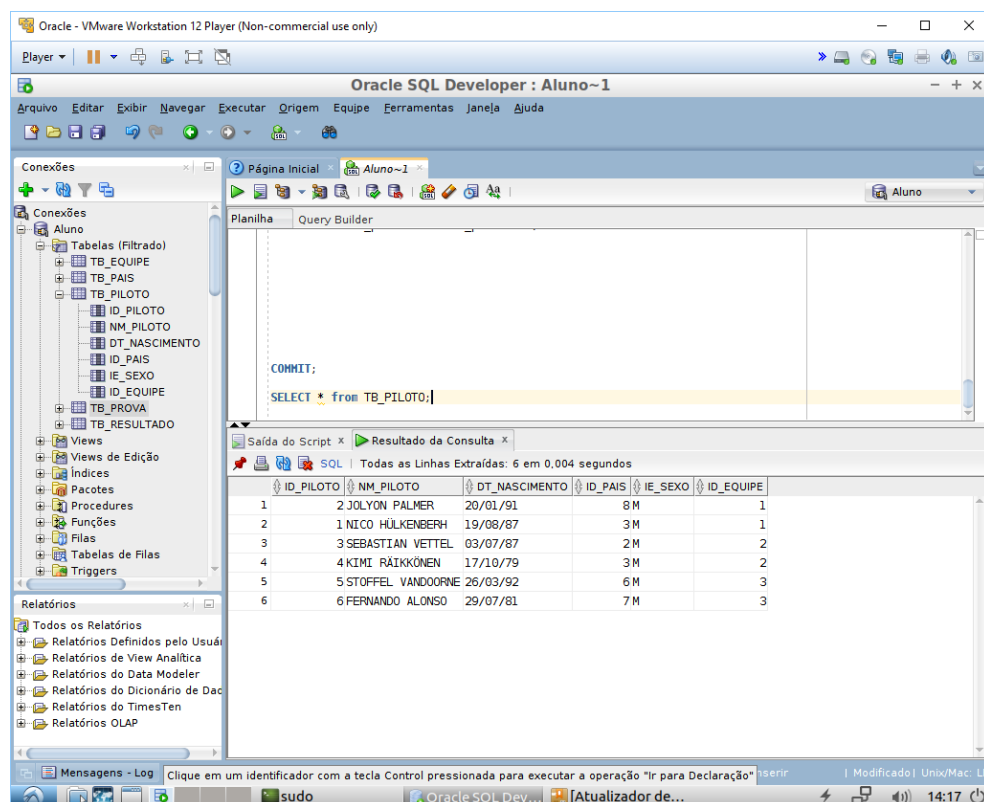
ROLLBACK;

Devemos notar que a linguagem SQL consegue implementar estas soluções, somente pelo fato de estar baseada em Banco de Dados, que garantem por si mesmo a integridade das relações existentes entre as tabelas e seus índices. Alguns SGBD aplicam COMMIT implícito a diversos comandos, de forma que você deve consultar no manual do SGBD quais são os comandos. Para exemplificar experimente os seguintes comandos;

COMMIT; --inicia um ponto de salvamento

SELECT * FROM TB_PILOTO;

A figura 05 apresenta o resultado da consulta a TB_PILOTO;



Oracle - VMware Workstation 12 Player (Non-commercial use only)

Oracle SQL Developer: Aluno~1

Planilha Query Builder

COMMIT;

SELECT * FROM TB_PILOTO;

Salida do Script x Resultado da Consulta x

Todas as Linhas Extraídas: 6 em 0,004 segundos

ID_PILOTO	NM_PILOTO	DT_NASCIMENTO	IE_SEXO	ID_EQUIPE
2	JOLYON PALMER	20/01/91	8 M	1
1	NICO HÜLKENBERG	19/08/87	3 M	1
3	SEBASTIAN VETTEL	03/07/87	2 M	2
4	KIMI RÄIKÖNEN	17/10/79	3 M	2
5	STOFFEL VANDORNE	26/03/92	6 M	3
6	FERNANDO ALONSO	29/07/81	7 M	3

Figura 05. Resultado da consulta SELECT * FROM TB_PILOTO

Para avaliar a capacidade de desfazer comandos DML utilize o código para inserir um novo piloto.

```
INSERT INTO TB_PILOTO VALUES (
    7, 'FELIPE MASSA', TO_DATE('25/04/1981', 'dd/mm/yyyy'), 1, 'M', 4
);
```

```
SELECT * FROM TB_PILOTO;
```

Observe como sai o resultado da consulta após a inserção realizada com sucesso na figura 06.

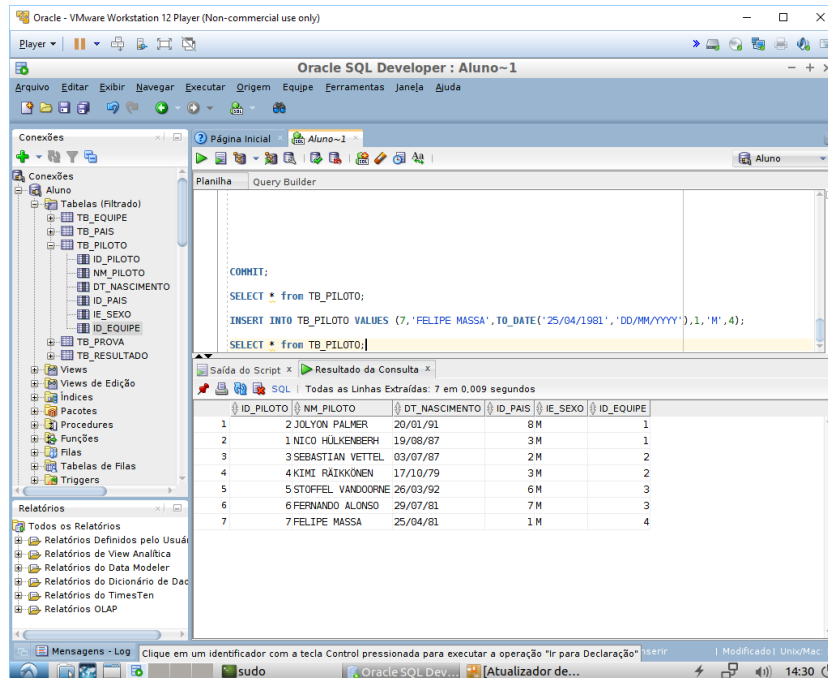


Figura 06. Resultado da consulta SELECT após o INSERT

Em seguida digite:

```
ROOLBACK;
```

```
SELECT * FROM TB_PILOTO;
```

Observe os dados que a seleção retorna após o comando ROOLBACK na figura 07.

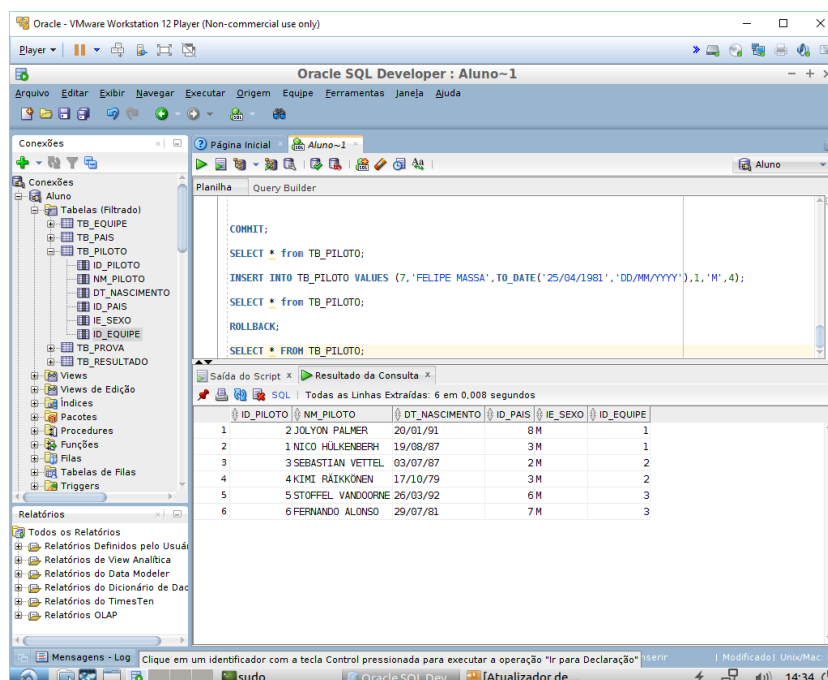


Figura 07. Retorno de dados após ROOLBACK

Os dados após o comando ROLLBACK voltam ao seu estado no último COMMIT válido, cancelando todas as operações de ocorrerem entre o COMMIT e o ROLLBACK. Se você acidentalmente esquecesse de colocar a cláusula WHERE em um comando DELETE você pode desfazer as ações. Experimente a sequência e avalie os resultados da SELECT.

```
COMMIT;
```

```
SELECT * FROM TB_PILOTO;
```

```
DELETE FROM TB_PILOTO; --Erro que apagará todas as linhas da tabela
```

```
SELECT * FROM TB_PILOTO;
```

```
ROLLBACK;
```

```
SELECT * FROM TB_PILOTO;
```

Os comandos COMMIT e ROLLBACK são capazes de efetivar ou cancelar tudo ou nada, não existe meio termo nestes comandos. Se for escrever um bloco com várias instruções a serem executadas, a série inteira pode ser salva ou desfeita como um grande grupo de comandos. A declaração **SAVEPOINT** faz parte dos comandos COMMIT e ROLLBACK. Este comando estabelece demarcações ou pontos dentro de uma transação. Seu objetivo é permitir que os comandos COMMIT ou ROLLBACK possam ser subdividir e que alguns pontos possam ser salvos ou ter a transação desfeita. Existem algumas regras para se utilizar o SAVEPOINT. A primeira é que todas as declarações SAVEPOINT deve incluir um nome. Cuidado para não duplicar os nomes dos SAVEPOINT, pois se você utilizar o mesmo nome a marcação, não acontece um erro e sim uma mudança do SAVEPOINT anterior para a nova posição, o que tornaria impossível recuperar e apagar aquele ponto. A terceira regra é que uma vez confirmada com COMMIT um grupo de transações todos os pontos de salvamento existentes serão apagados da memória e quaisquer referências a eles após o COMMIT implicará em erro. A sintaxe do comando é simples e segue a estrutura:

```
SAVEPOINT nome_ponto;
```

Para exemplificar experimente:

```
SAVEPOINT sp_1;
```

```
INSERT INTO TB_PILOTO VALUES (  
    7, 'FELIPE MASSA', TO_DATE('25/04/1981', 'dd/mm/yyyy'), 1, 'M', 4  
);
```

```
SELECT * FROM TB_PILOTO;
```

```
SAVEPOINT sp_2;
```

```
INSERT INTO TB_PILOTO VALUES (  
    8, 'LANCE STROLL', TO_DATE('29/10/1990', 'dd/mm/yyyy'), 12, 'M', 4  
);
```

```
SELECT * FROM TB_PILOTO;
```

```
ROLLBACK TO sp_2;
```

```
SELECT * FROM TB_PILOTO;
```

O SAVEPOINT é útil na gestão de uma grande série de transações em que a validação incremental é exigida. Pode lhe ajudar a corrigir e validar erros em partes das instruções antes de validar todas as ações.

Síntese

Nesta sexta aula, você aprendeu como alterar tabelas com o comando SQL ALTER TABLE. Viu como funciona a inserção de dados com o INSERT INTO. Aprendeu a alterar e excluir registros de dados com UPDATE e DELETE. Vimos a sintaxe completa da instrução SELECT e você pode experimentar os comandos básicos da SELECT. Além disso você pode aprender sobre os operadores BETWEEN, IN, LIKE e IS NULL.

Atividade

1. Com base na tabela gere o script que insere os dados da TB_EQUIPE

Equipe	
Ferrari	Itália
Force India-Mercedes	Índia
Haas-Ferrari	Estados Unidos
McLaren-Honda	Reino Unido
Mercedes	Alemanha
Red Bull-TAG Heuer	Austria
Renault	França
Sauber-Ferrari	Suíça
Toro Rosso ^[26]	Itália
Williams-Mercedes	Reino Unido

2. Faça uma pesquisa e complete a TB_PILOTO com os pilotos do calendário de 2017.
-

Glossário

ALTER TABLE – comando SQL que permite alterações na estrutura de uma tabela;

ADD – adiciona colunas ou restrições em uma tabela;

MODIFY – altera as características de uma tabela;

DROP – exclui colunas ou restrições em uma tabela;

DESC – descreve a estrutura de uma tabela;

INSERT INTO – comando SQL para inserir dados em uma tabela;

VALUES – apresenta os valores ;

NULL – instrução para informar valor nulo ;

INSERT ALL – inserção de múltiplas linhas;

AS – apelido que pode ser atribuído a uma coluna ou ao nome de uma tabela;

TO_DATE – instrução que converte string em data;

AND – operador E, utilizado em operações lógicas

OR – operador OU, utilizado em operações lógicas

BETWEEN – entre dois valores

IN – indica uma lista de valores

LIKE – compara um parâmetro alfanumérico

IS NULL – valida se é nulo

EXISTS – valida se existe

ORDER BY – coloca ou ordena uma lista

DISTINCT – retorna dados distintos

UPDATE – comando para atualizar dados

DELETE – comando para apagar dados

CREATE TABLE AS – cria uma tabela com base em uma consulta

RENAME – permite trocar o nome de uma tabela

COMMIT – grava todas as alterações

ROLLBACK – desfaz quaisquer alterações

SAVEPOINT – pontos ou marcações dentro de uma transação