

# TUTORIAL: SQL

---

Banco de Dados Relacional

# SQL

O SQL é uma linguagem padrão para manipulação de registros em bancos de dados relacionais. A sigla SQL vem dos termos em inglês Structured Query Language, que podem ser traduzidos para o português como Linguagem de Consulta Estruturada.

## Trabalhando com SQL

Para utilizar o sql, entre no site <https://livesql.oracle.com>:



Clique em Sign In para criar sua conta:



Crie sua conta clicando em Criar Conta:



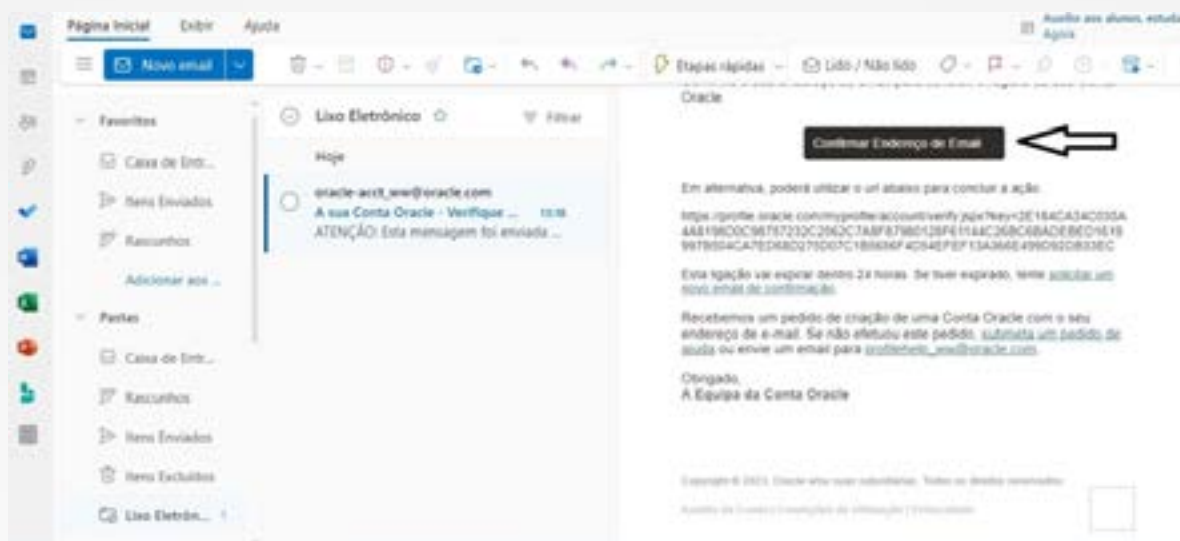
Preencha o formulário e clique em Criar Conta:



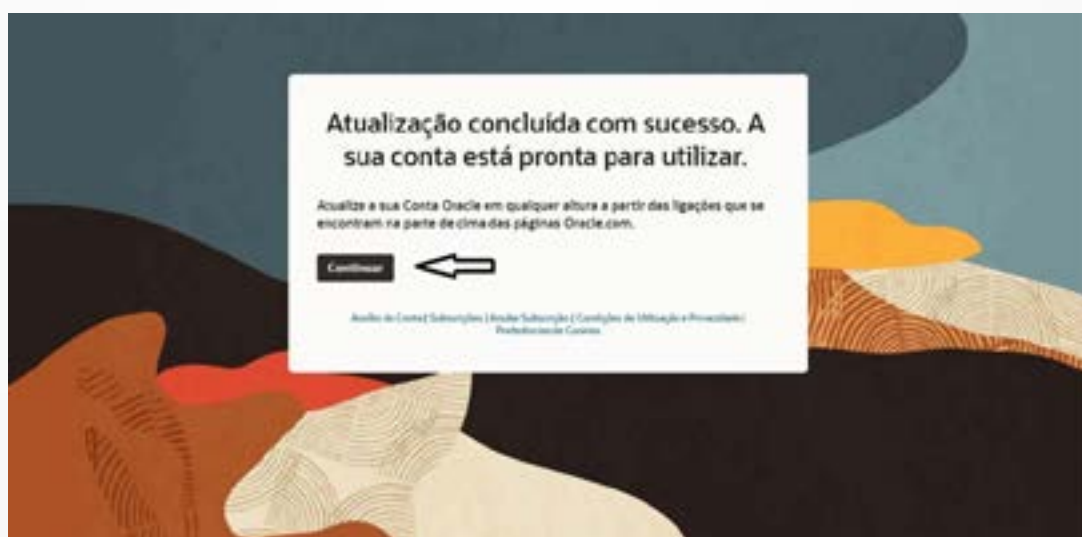
Será enviado e-mail de confirmação, após preenchimento e criação da conta:



**OBS:** Entre no seu e-mail institucional para finalizar a criação da conta. Verifique se o e-mail foi enviado para lixeira. Isso pode ocorrer com frequência.



**OBS:** Clique em Confirmar Endereço de E-mail e Continuar.

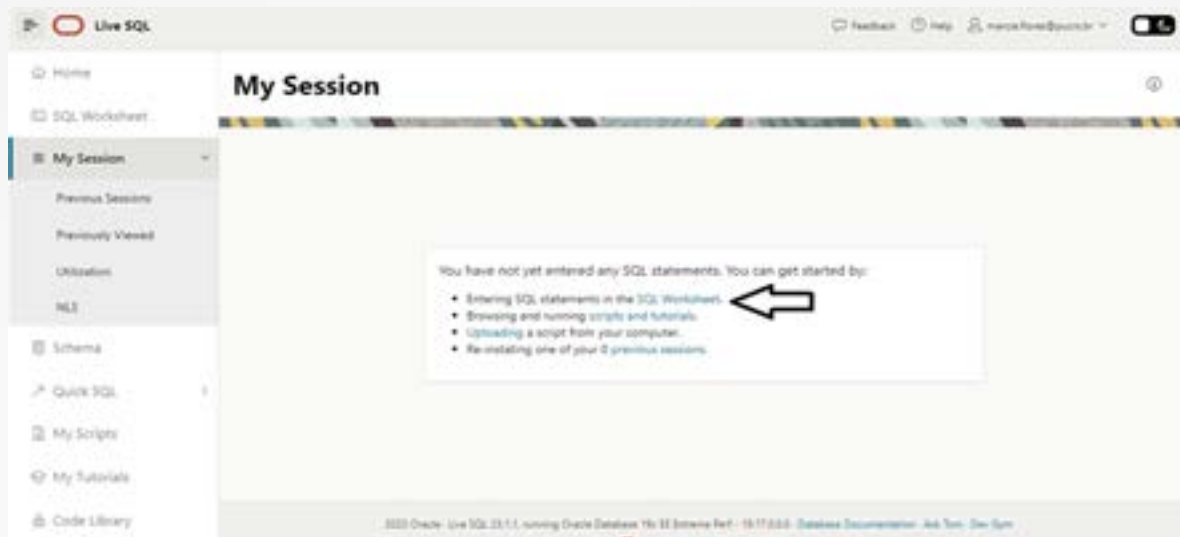


Com a conta criada, verificada e confirmada, entre com seus dados de login e clique em Iniciar sessão:



**IMPORTANTE:** Se, por acaso, ao realizar o login, for apresentada a mensagem **“Sua sessão foi finalizada”**, na mesma tela, no navegador, clique no cadeado, clique em cookies, remova os cookies do livesql, dê enter e aceite os termos.

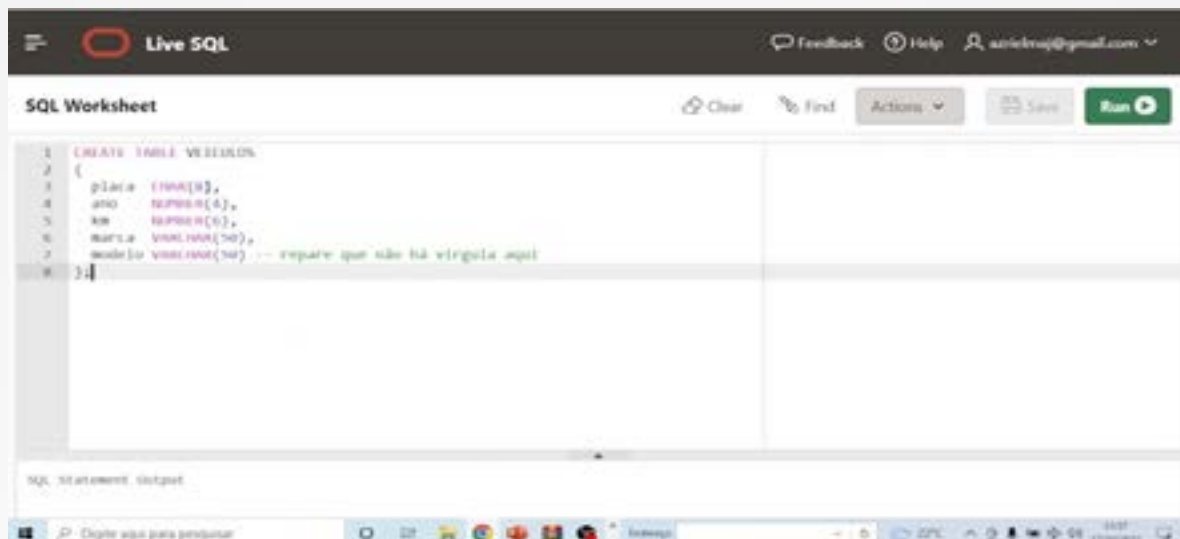
Para iniciar uma sessão, clique em My Session → SQL Worksheet:



Agora, com a conta criada e já logado ao [livesql.oracle.com](https://livesql.oracle.com), vamos iniciar a criação do banco de dados, tabelas, inserção de dados, alterações, deleção e consultas:



Para criar tabelas, no livesql, insira o código e clique em Run, para que a tabela seja criada:



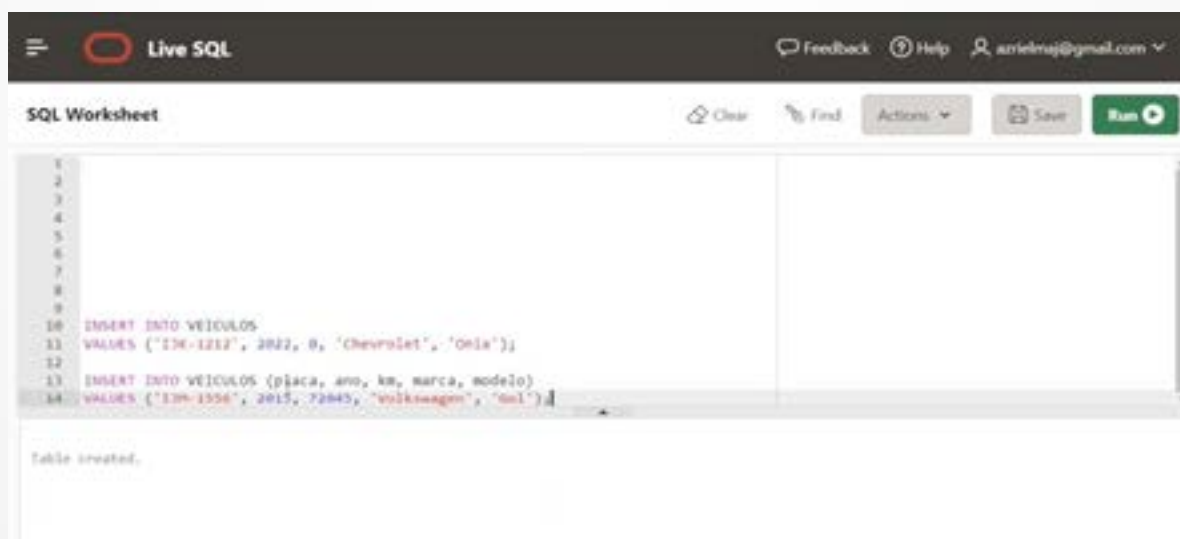
The screenshot shows the Live SQL web application. The SQL Worksheet area contains the following code:

```
1 CREATE TABLE VEICULOS
2 {
3   placa VARCHAR(10),
4   ano NUMBER(4),
5   km NUMBER(6),
6   marca VARCHAR(50),
7   modelo VARCHAR(50) -- repare que não há vírgula aqui
8 }
```

The interface includes a top bar with the Live SQL logo, a user profile, and navigation links. Below the code editor are buttons for Clear, Find, Actions, Save, and Run. The bottom section is labeled 'SQL statement output'.

Observe que, para criar uma tabela, utilizamos o CREATE TABLE NomeDaTabela, abre parênteses, e os atributos oriundos do Diagrama Entidade Relacionamento (DER), fecha parentes. Todos atributos devem conter um nome, por exemplo: placa e o tipo de dados que esse atributo receberá.

Para inserir dados na tabela criada, insira o código, selecione as linhas a serem inseridas e clique em Run:



The screenshot shows the Live SQL web application with the following code in the SQL Worksheet:

```
10 INSERT INTO VEICULOS
11 VALUES ('13E-1212', 2022, 0, 'Chevrolet', 'Onix');
12
13 INSERT INTO VEICULOS (placa, ano, km, marca, modelo)
14 VALUES ('13M-1556', 2015, 72845, 'Volkswagen', 'Gol');
```

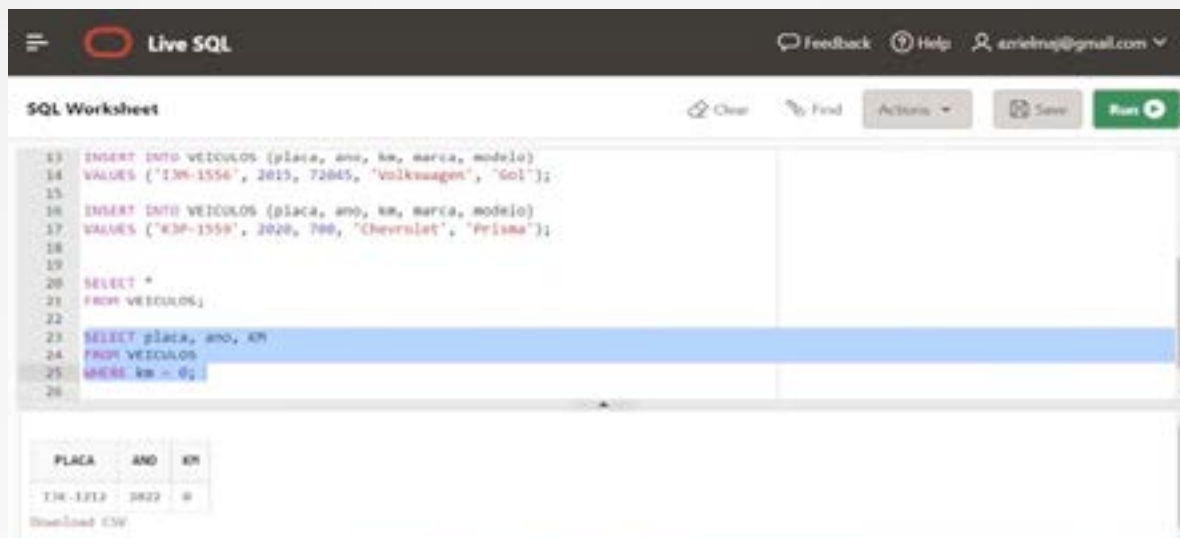
The interface shows the same top bar and buttons as the previous screenshot. The bottom section now displays the message 'Table created.'

Ao inserir dados na tabela, utilize o INSERT INTO NomeDaTabela. Para isso, podemos inserir diretamente os valores, em ordem, conforme criação do Banco de dados, utilizando o VALUES e os valores, entre parênteses, conforme a imagem acima. É possível, ainda, inserir os nomes dos atributos entre parênteses e, logo após, inserir o VALUES, com os valores, entre parênteses, seguindo a ordem dos atributos, que aparecem no INSERT INTO, conforme imagem.

Para visualizar os dados inseridos, utilize Select \* From Veiculos. Serão apresentados os dados inseridos na tabela Veículos.



Para criar filtros com a cláusula Select, insira o código e clique em Run:



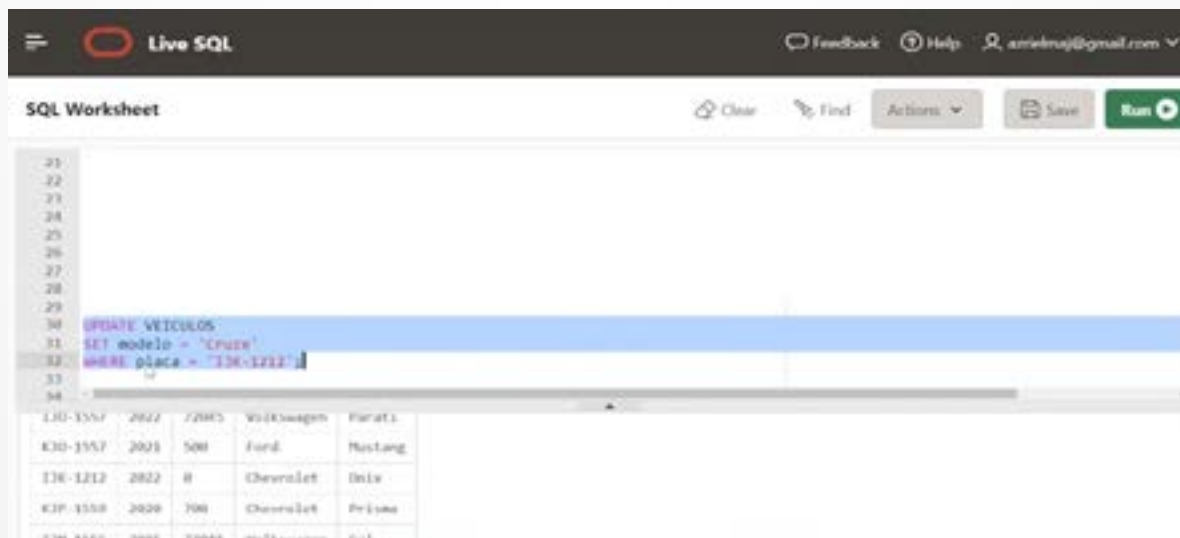
```
13 INSERT INTO VEICULOS (placa, ano, km, marca, modelo)
14 VALUES ('IJK-1556', 2015, 72045, 'Volkswagen', 'Gol');
15
16 INSERT INTO VEICULOS (placa, ano, km, marca, modelo)
17 VALUES ('KJP-1559', 2020, 700, 'Chevrolet', 'Prisma');
18
19
20 SELECT *
21 FROM VEICULOS;
22
23 SELECT placa, ano, km
24 FROM VEICULOS
25 WHERE km = 0;
```

PLACA	ANO	KM
IJK-1556	2015	0

Download CSV

O Select, nessa consulta, apresentará a placa, o ano e a KM (veja que todos os atributos que estão entre o Select e o From são, justamente, os que serão apresentados na tabela). Os dados vêm da tabela Veiculos, devido ao uso da cláusula From, para inserirmos uma condição para a filtragem das linhas sendo consultadas da tabela; logo, somente as linhas cujos veículos possuam a kilometragem com valor zero serão retornadas no resultado.

Para atualizar tabela, utilize o código e clique em Run:



```
30
31
32 UPDATE VEICULOS
33 SET modelo = 'Cruze'
34 WHERE placa = 'IJK-1212';
```

IJK-1556	2015	72045	Volkswagen	Gol
KJP-1559	2020	700	Ford	Mustang
IJK-1212	2022	0	Chevrolet	Onix
KJP-1559	2020	700	Chevrolet	Prisma
IJK-1556	2015	72045	Volkswagen	Gol

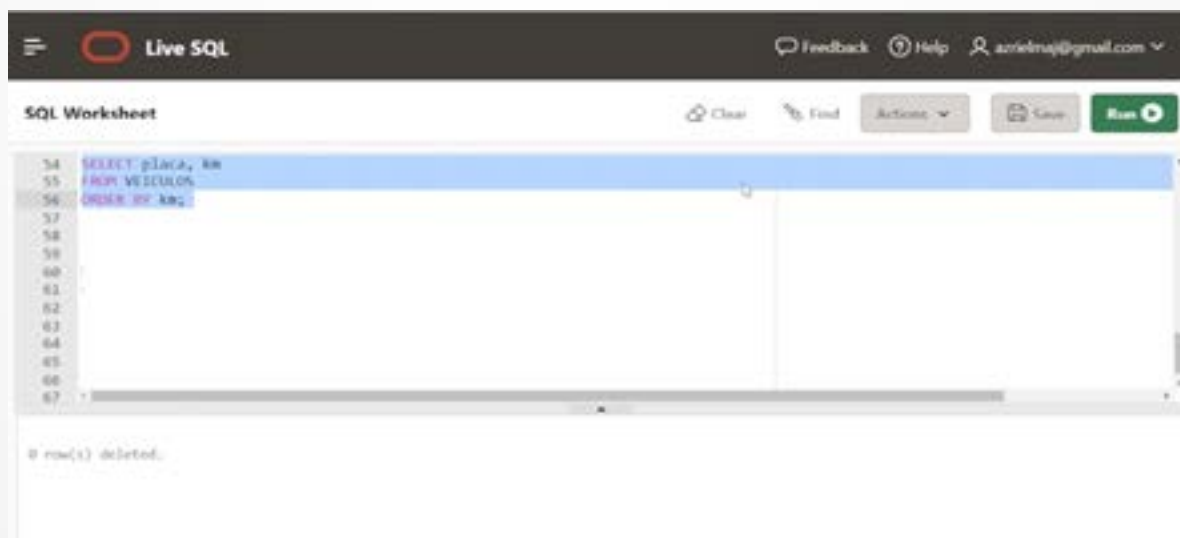
O UPDATE faz a alteração nos dados existentes na tabela. Para isso, utilizamos o SET, atributo a ser alterado, e a cláusula WHERE, que insere a condição para que a alteração seja realizada. No caso, a alteração ocorrerá na linha onde temos a placa IJK-1212. Antes o modelo era Onix, com o Update, será atualizado para Cruze. Para confirmar a alteração na tabela, utilize o Select \* From Veiculos, para visualizar a atualização.

Para deletar dados de uma tabela, utilize o código e clique em Run:



O DELETE apagará apenas as linhas indicadas na cláusula Where, sem risco de perder a tabela. Para isso, é necessário setar, com o uso do From, a tabela que terá sua linha deletada, no caso Veiculos. Na condição WHERE, a linha que será apagada é a que contém a placa igual a= IJM-1556. Para confirmar a deleção na tabela, utilize o Select \* From Veiculos, para visualizar o registro, agora inexistente, na tabela.

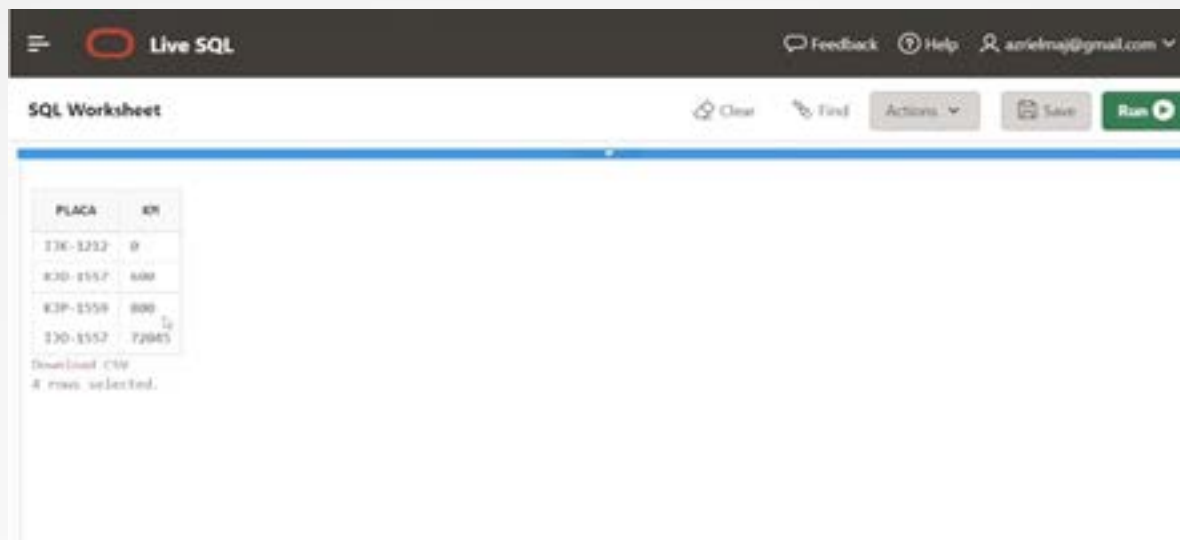
Para ordenar os dados de uma tabela, utilize o código e clique em Run:



O ORDER BY ordenará placa e Km da tabela Veiculos em ordem ASC, ou ascendente, visto que nada foi inserido após Km. Ela também pode ser ordenada de forma DESC, ou descendente. Nesse caso, é necessário incluir no filtro a palavra DESC, após Km.



O resultado da consulta, aplicando ORDER BY, é o que segue:

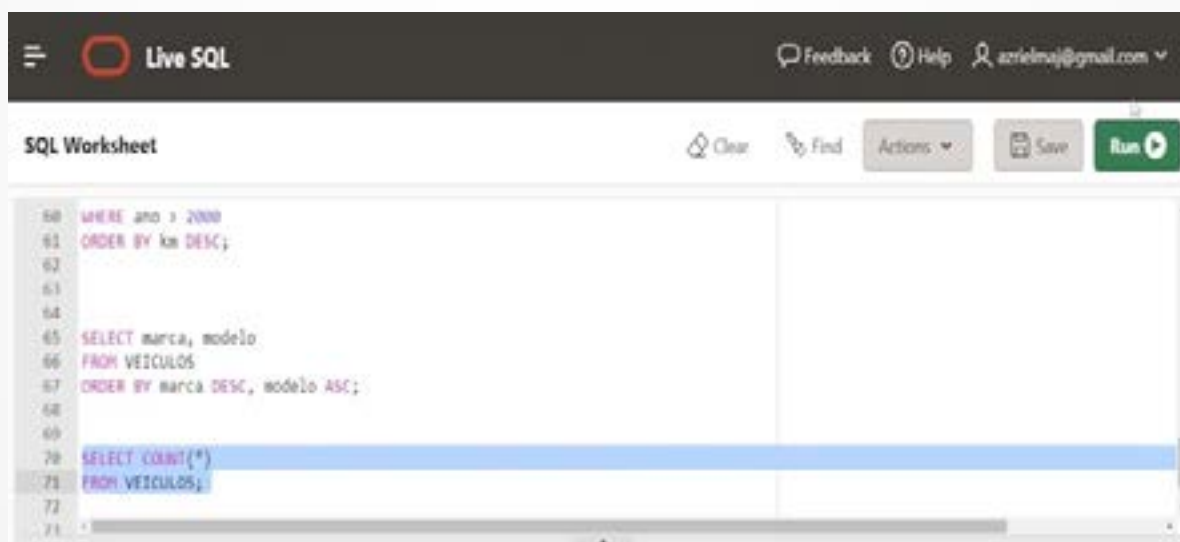


PLACA	KM
13E-1232	0
K3D-1557	600
K3P-1559	800
13D-1557	72045

Download CSV  
4 rows selected.

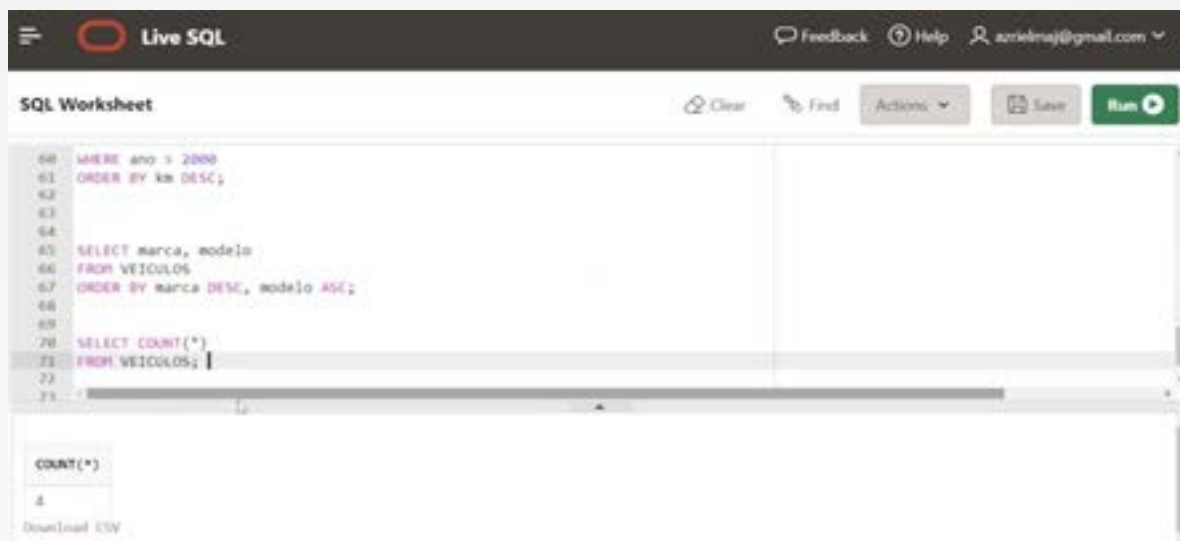
Observe que a tabela traz apenas a placa e a km (atributos entre o Select e From) e elas estão ordenadas de forma ascendente devido a condição da cláusula Where.

O COUNT é uma função de agregação que serve para verificarmos quantas linhas existem em uma tabela. Assim, para contar as linhas de uma tabela, utilize o código e clique em Run:



```
60 WHERE ano > 2000
61 ORDER BY km DESC;
62
63
64
65 SELECT marca, modelo
66 FROM VEICULOS
67 ORDER BY marca DESC, modelo ASC;
68
69
70 SELECT COUNT(*)
71 FROM VEICULOS;
```

O COUNT contará e apresentará o número de linhas utilizadas na tabela Veiculos. Assim:



The screenshot shows the 'Live SQL' web application interface. The 'SQL Worksheet' area contains the following SQL code:

```
60 WHERE ano > 2000  
61 ORDER BY km DESC;  
62  
63  
64  
65 SELECT marca, modelo  
66 FROM VEICULOS  
67 ORDER BY marca DESC, modelo ASC;  
68  
69  
70 SELECT COUNT(*)  
71 FROM VEICULOS;  
72  
73
```

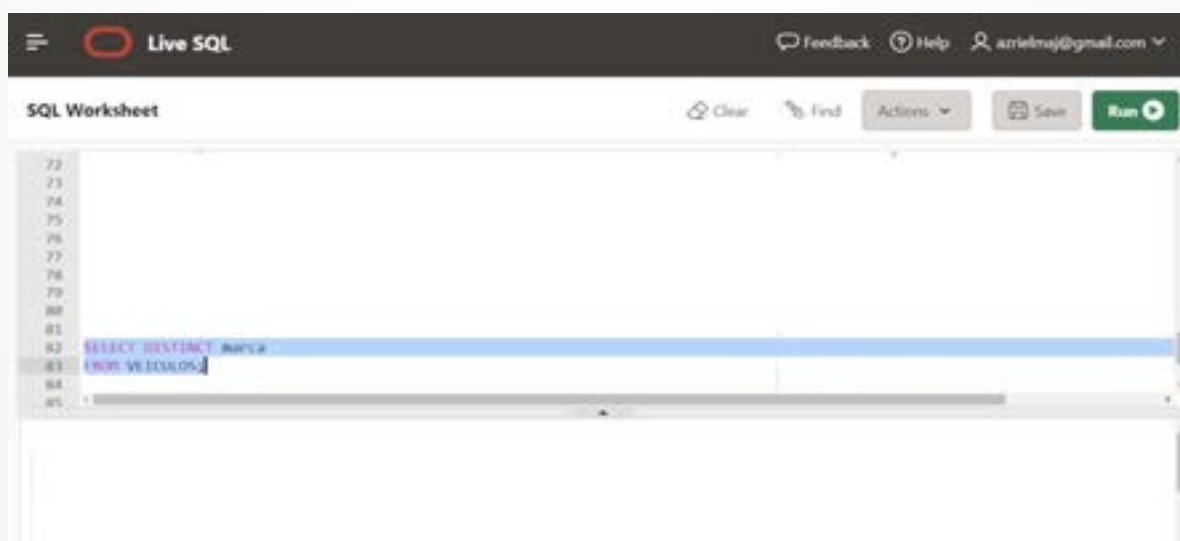
Below the code editor, the result of the last query is displayed in a table:

COUNT(*)
4

A 'Download CSV' link is visible at the bottom left of the results area.

Verificamos, com o Count, que a tabela Veiculos possui 4 registros. Para confirmar o número de registros na tabela, utilize o Select \* From Veiculos, para visualizá-los.

O DISTINCT é usado para trazer nas consultas onde há mais de uma ocorrência do mesmo dado ou esse dado apenas uma vez. Para trazer as ocorrências de dados apenas uma vez, use o código e clique em Run:

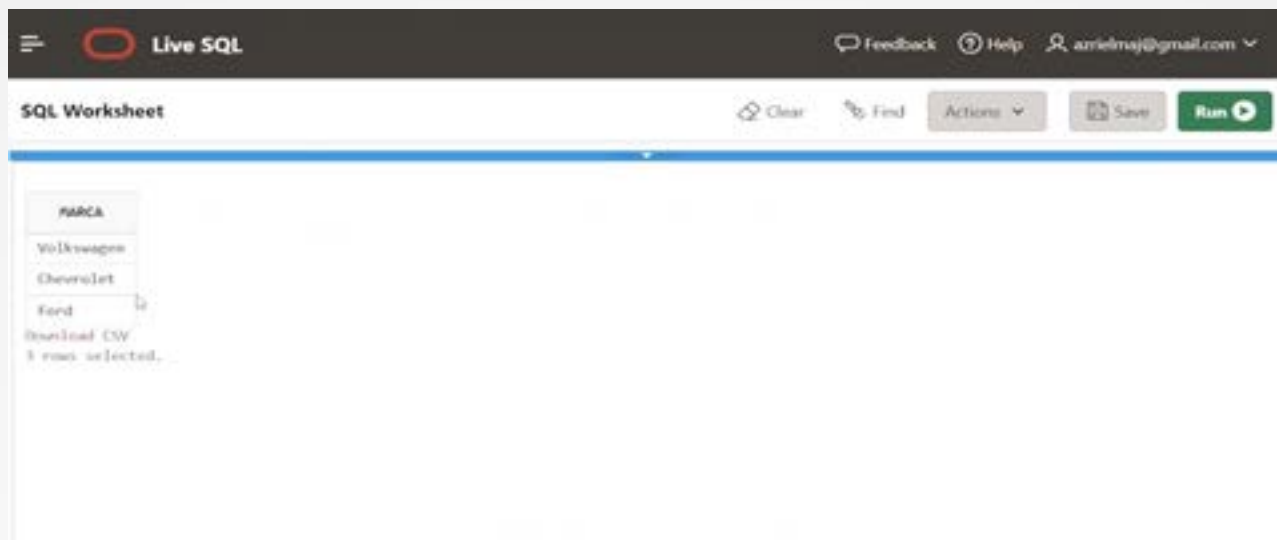


The screenshot shows the 'Live SQL' web application interface. The 'SQL Worksheet' area contains the following SQL code:

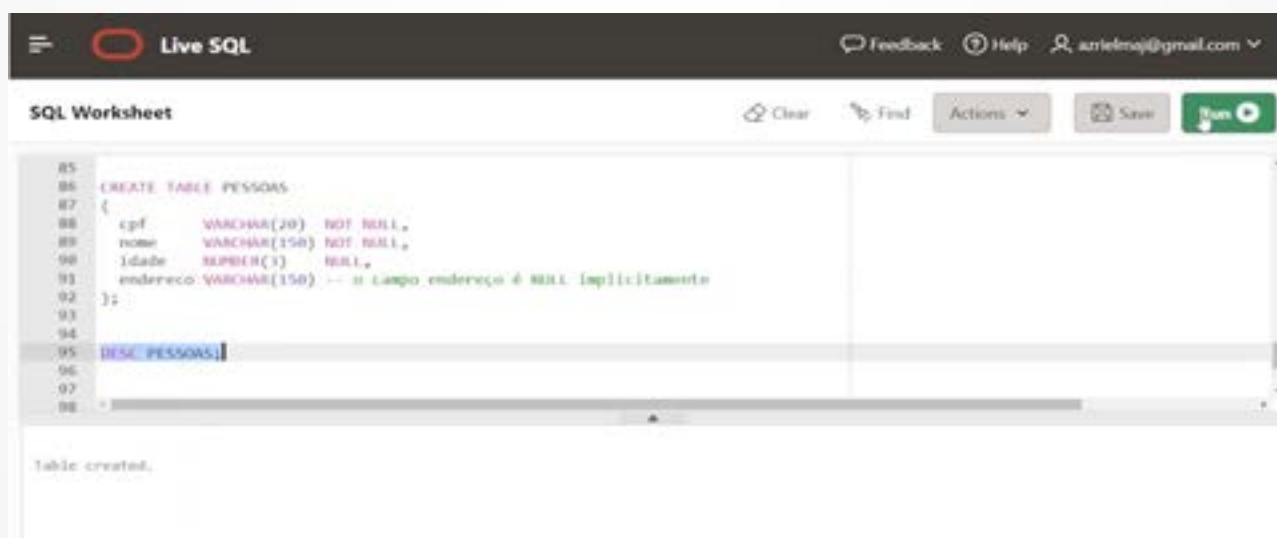
```
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82 SELECT DISTINCT marca  
83 FROM VEICULOS;  
84  
85
```

The code is currently selected, and the 'Run' button is visible in the top right corner of the worksheet area.

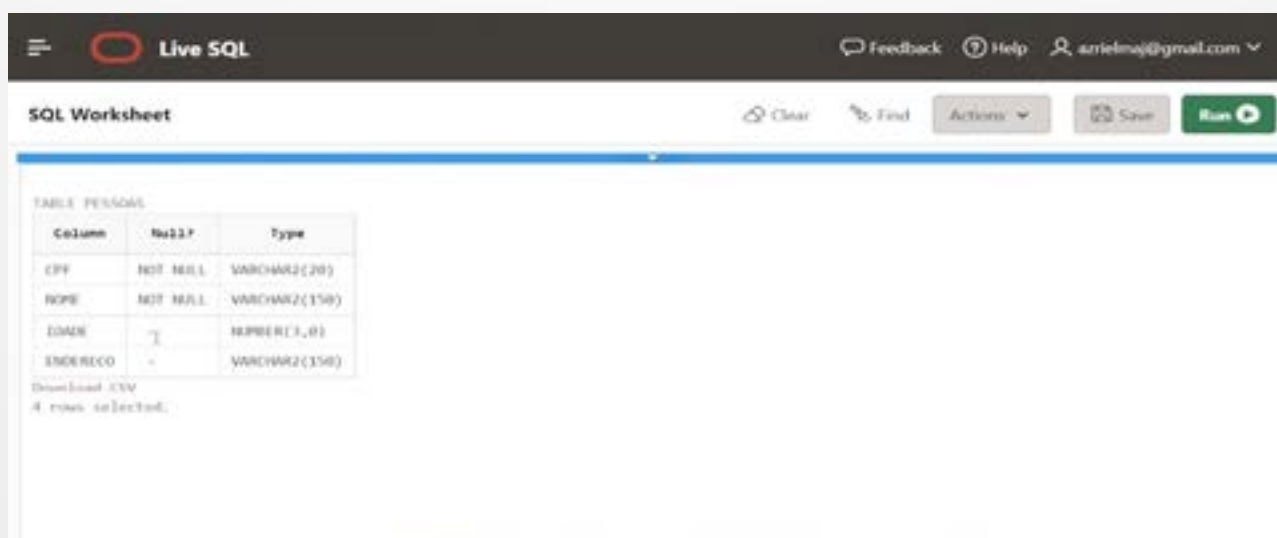
O resultado do filtro, utilizando o Distinct, é o que segue:



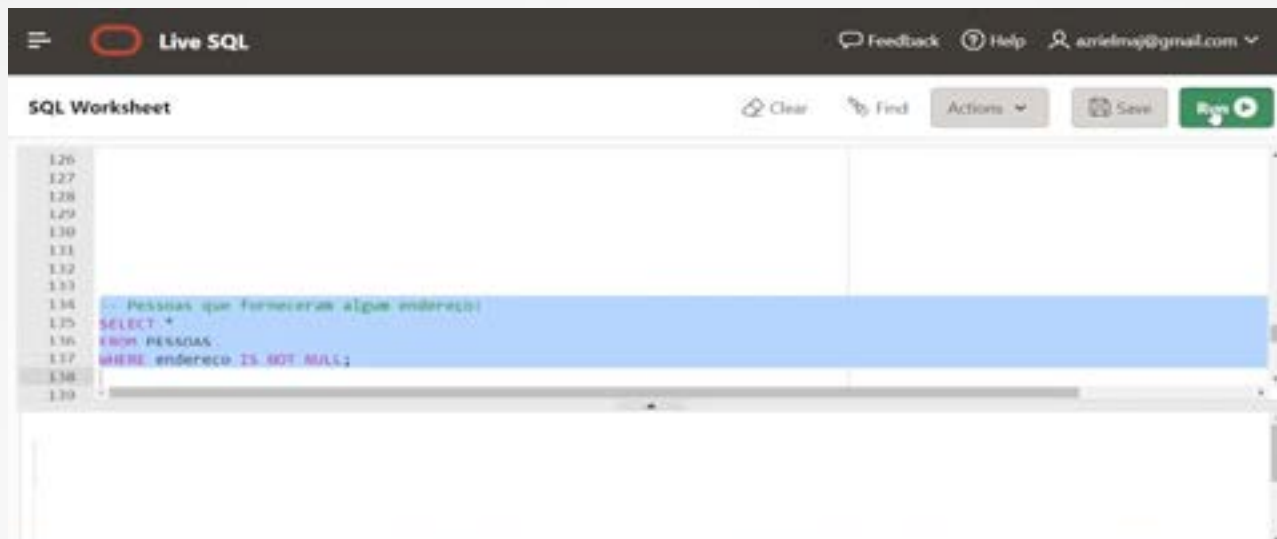
Agora, para seguirmos com as consultas, é necessário criar a tabela pessoas, com o Create Table NomeDaTabela e seus atributos, como já foi feito. Para verificar as colunas da tabela criada, utilize o comando DESC NomeDaTabela:



O resultado apresentado, com o uso do DESC, será o que segue, resultando na estrutura da tabela:



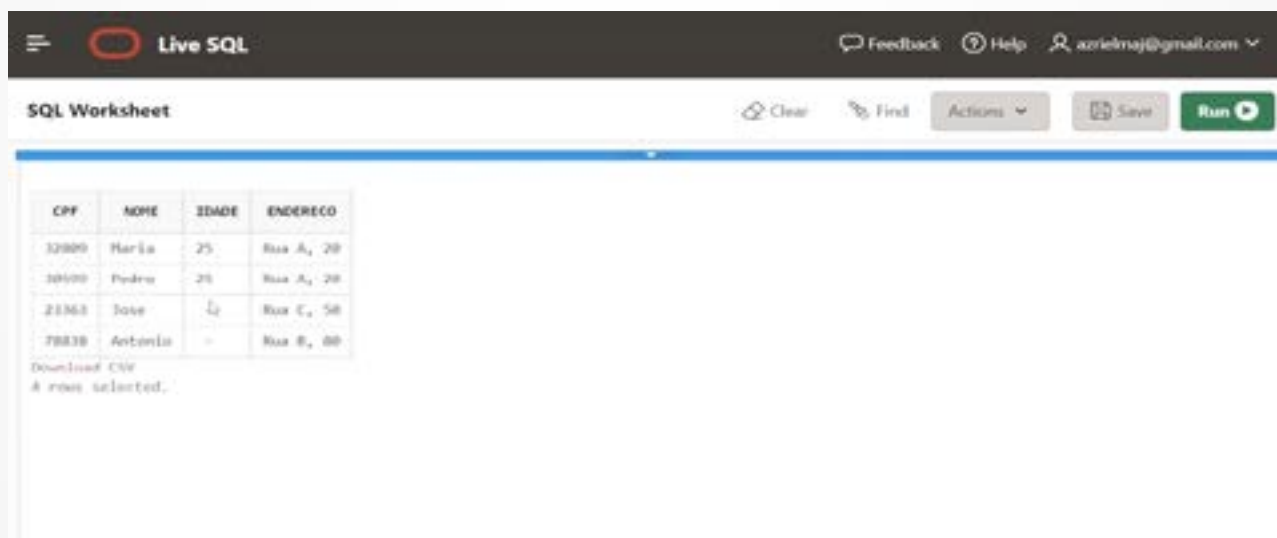
Os comandos IS NOT NULL e NULL são usados para filtros onde se quer verificar a presença de tabelas com colunas sem dados e com dados. Para isso, uso o código a seguir e clique em Run:



The screenshot shows the 'Live SQL' web application. The 'SQL Worksheet' area contains the following SQL query:

```
-- Pessoas que forneceram algum endereço;  
SELECT *  
FROM PESSOAS  
WHERE endereco IS NOT NULL;
```

O Is Not Null trará a tabela, com a coluna endereço preenchida, mesmo se as demais colunas estiverem sem dados:

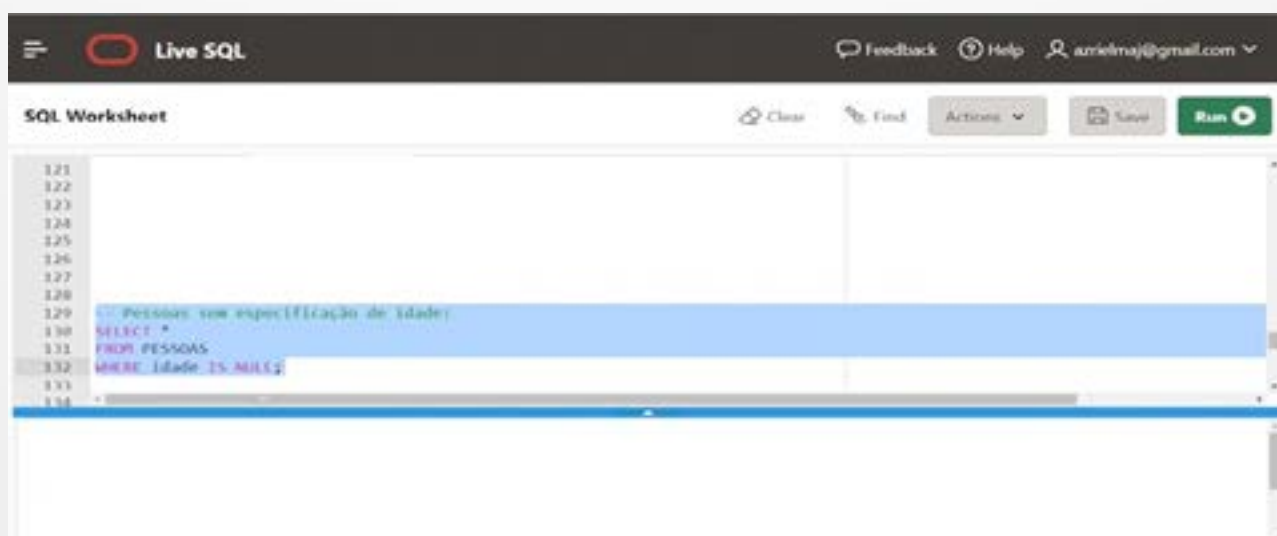


The screenshot shows the results of the SQL query in the 'Live SQL' interface. The results are displayed in a table with the following data:

CPF	NOME	IDADE	ENDEREÇO
32989	Maria	25	Rua A, 20
30500	Pedro	20	Rua A, 20
21363	João	42	Rua C, 50
78838	Antônio	-	Rua B, 80

Download CSV  
A row selected.

Já o comando Is null trará a tabela com a coluna selecionada, apenas sem dados. As demais colunas que contêm dados também serão mostradas:



The screenshot shows the 'Live SQL' web application. The 'SQL Worksheet' area contains the following SQL query:

```
-- Pessoas sem especificação de idade;  
SELECT *  
FROM PESSOAS  
WHERE idade IS NULL;
```

A tabela que será mostrada traz a coluna idade, sem dados, e as demais colunas da tabela com dados:

CPF	NOME	IDADE	ENDEREÇO
00038	Ana Paula	-	-
23363	Jose	-	Rua C, 50
78838	Antonio	-	Rua B, 80
29385	Carlos	-	-

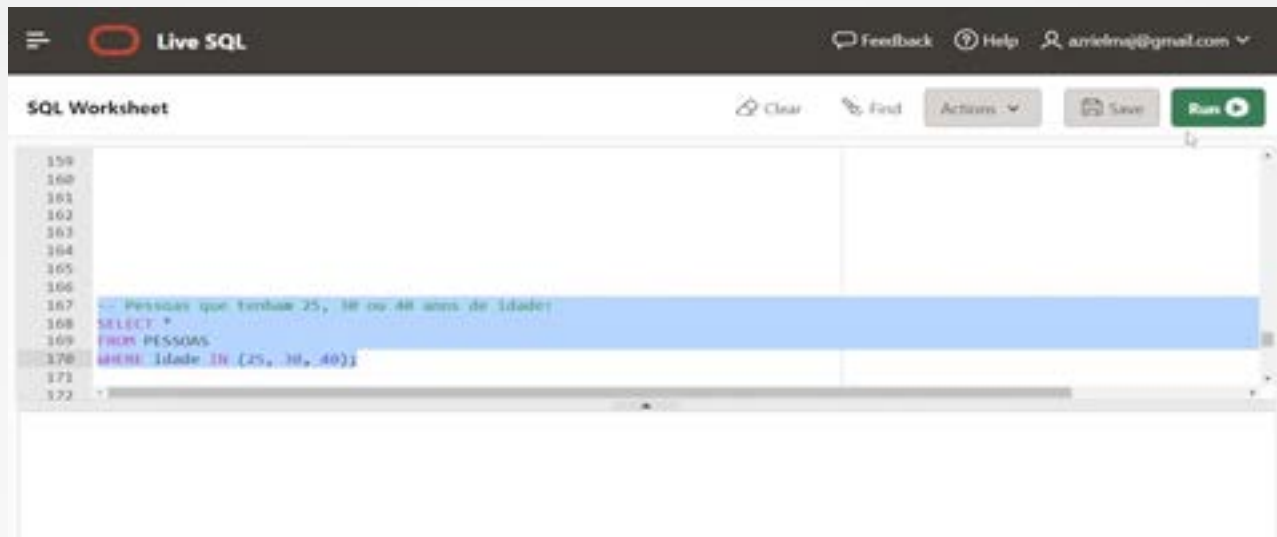
Download CSV  
4 rows selected.

Os operadores LIKE e IN: o operador Like é utilizado para localizar textos. Para isso, utilizados o “%”. O “%” substitui zero ou mais caracteres. Para visualizar o comando, utilize o código abaixo e clique em Run:

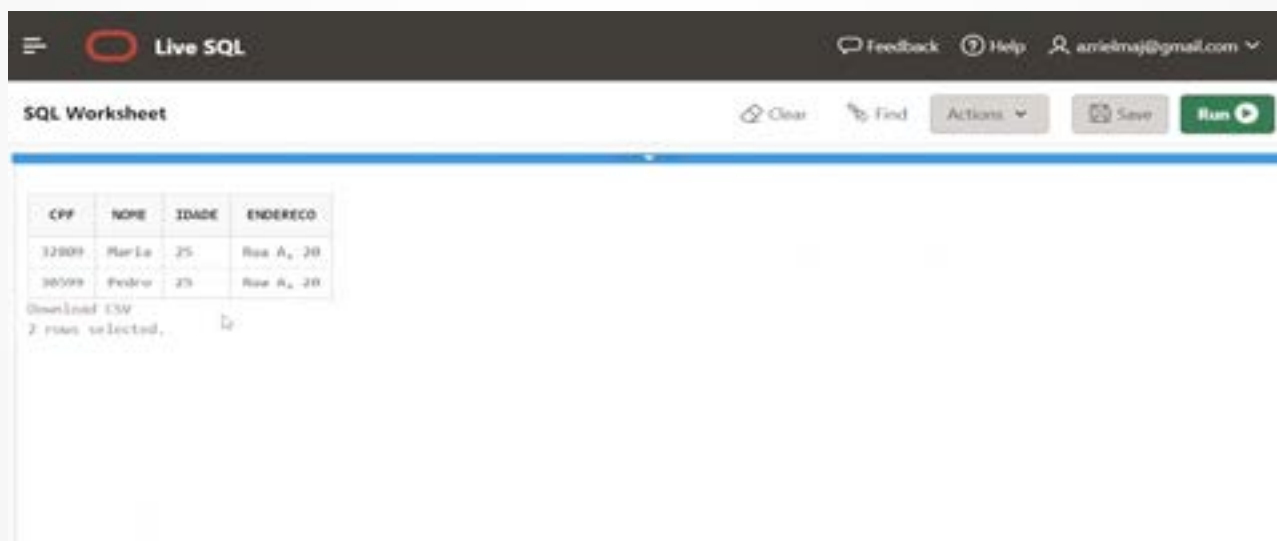
```
139
140 -- Pessoas com nomes iniciando com a letra 'A':
141 SELECT *
142 FROM PESSOAS
143 WHERE nome LIKE 'A%';
144
145
146
147
148
149
150
151
152
```

O resultado dessa consulta trará, na coluna nome, todas as pessoas cujo nome inicie com “A”.

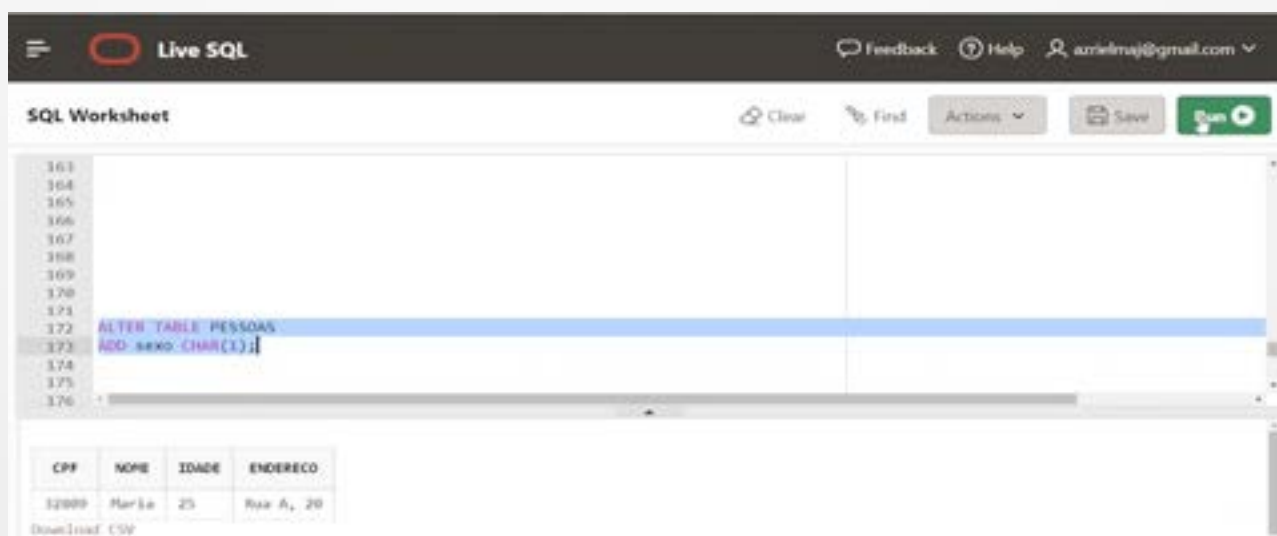
O operador IN corresponde aos valores que estão numa lista:



O resultado da consulta trará as pessoas com idades de 25, 30 e/ou 40 anos. Como no banco de dados, não temos pessoas com 30 e 40 anos, o resultado será o que segue:

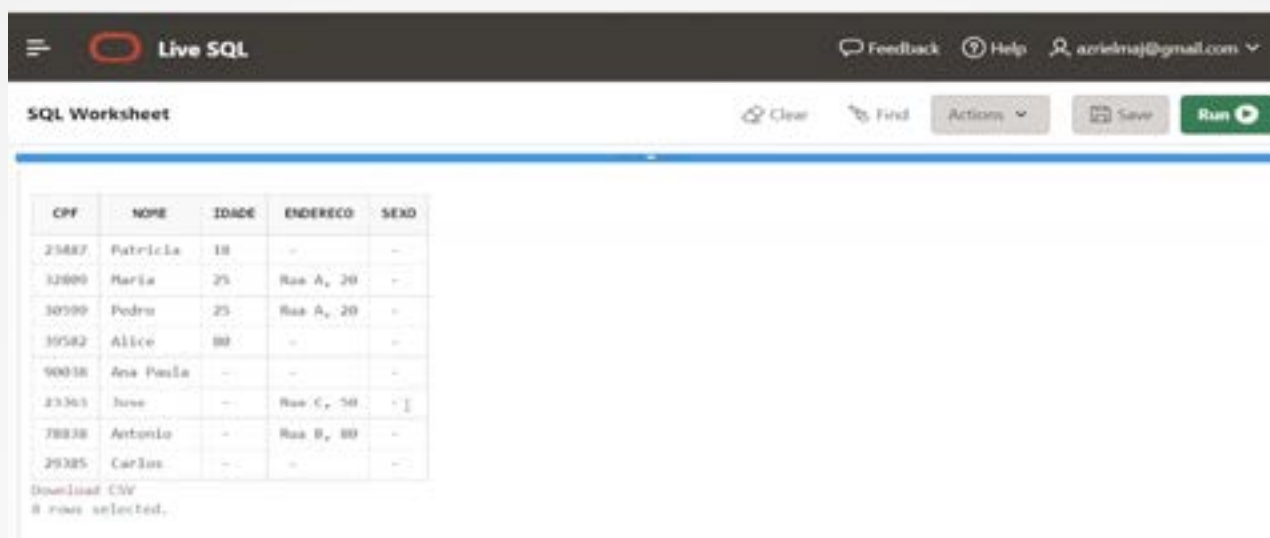


O comando ALTER TABLE é utilizado para alterar a estrutura da tabela, ou seja, com esse comando, pode-se incluir novas colunas ou modificá-las:





O resultado do comando de alteração pode ser visto utilizando o comando DESC Pessoas ou Select \* From Pessoas, assim:



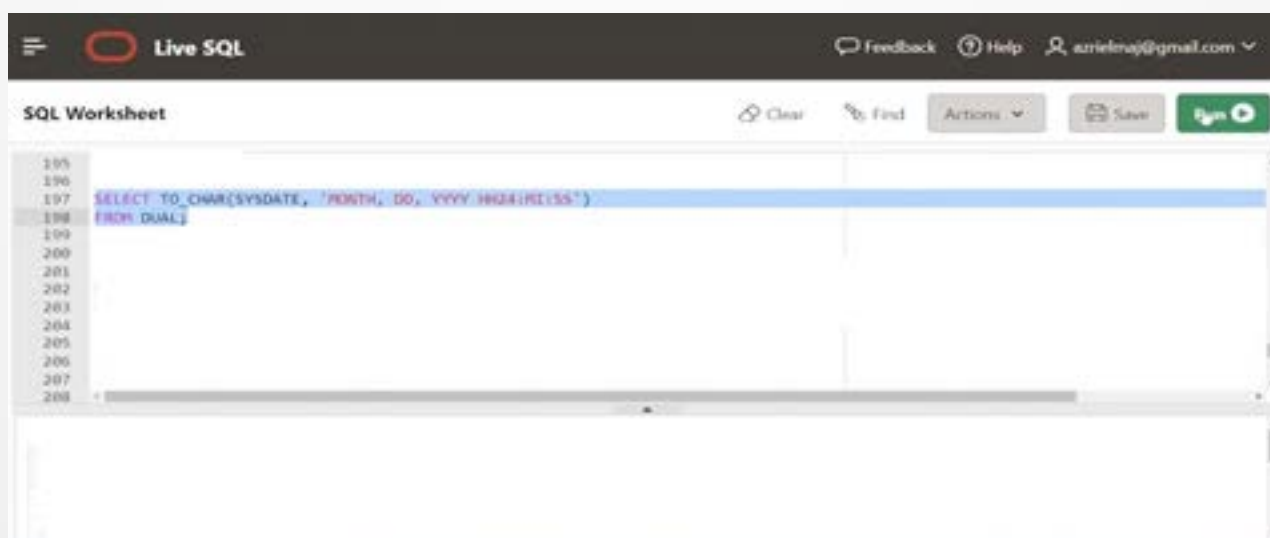
CPF	NOME	IDADE	ENDEREÇO	SEXO
234567	Patricia	18	-	-
32000	Maria	25	Rua A, 20	-
30500	Pedro	25	Rua A, 20	-
30582	Alice	80	-	-
90038	Ana Paula	-	-	-
23363	João	-	Rua C, 50	-
78838	Antonio	-	Rua B, 80	-
29325	Carlos	-	-	-

Download CSV  
0 rows selected.

Observe que a coluna sexo foi adicionada e que nela não há dados, devido sua alteração recente.

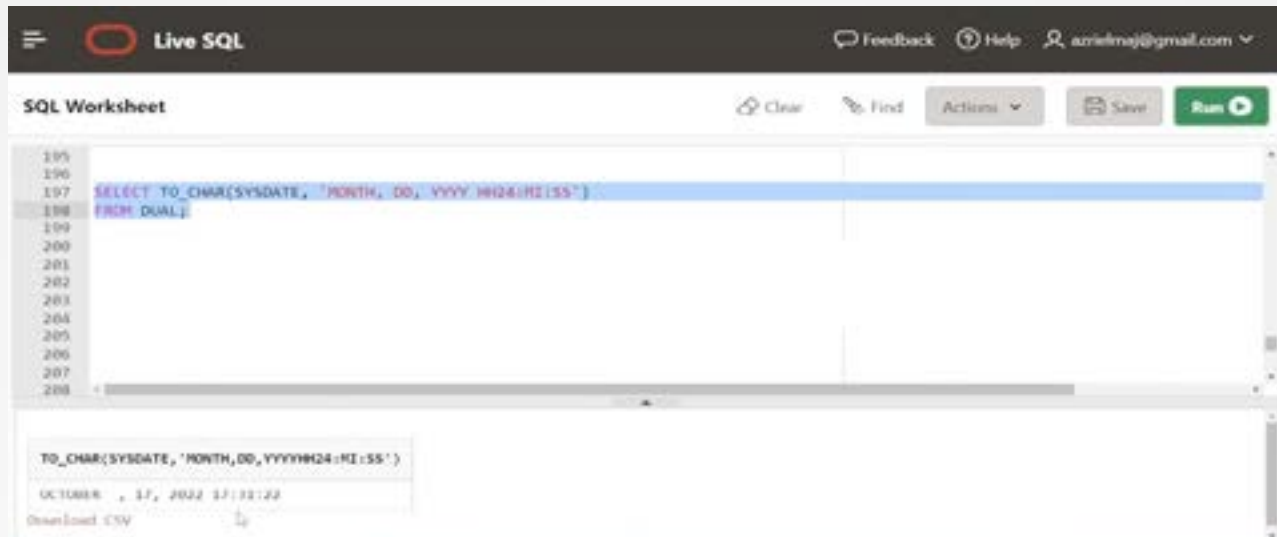
**IMPORTANTE:** Para deletar uma coluna da tabela, utilize o comando ALTER TABLE Pessoas DROP COLUMN idade. O resultado do filtro será a deleção da coluna idade da tabela Pessoas.

O DATE é utilizado em campos, cujo preenchimento deve conter o dia (dd), mês (mm), ano (aaaa) e hora no formato de 24h, no formato de minutos e segundos. Para realizar uma nova consulta, remova a coluna idade e adicione a coluna dataNasc com tipo de dado Date e Null. Será necessário fazer Insert Into na tabela Pessoas para incluir datas de nascimento. Assim, podemos seguir com a construção da consulta, usando To\_Char, para conversão da Data, e Sysdate, para trazer a data e hora do Sistema Operacional (S.O.). A tabela Dual é uma tabela interna Oracle, justamente usada para esses tipos de casos:



```
195
196
197 SELECT TO_CHAR(SYSDATE, 'MONTH, DO, YYYY HH24:MI:SS')
198 FROM DUAL;
```

O resultado da consulta traz a data do S.O.:



The screenshot shows the 'Live SQL' web application. The SQL Worksheet area contains the following query:

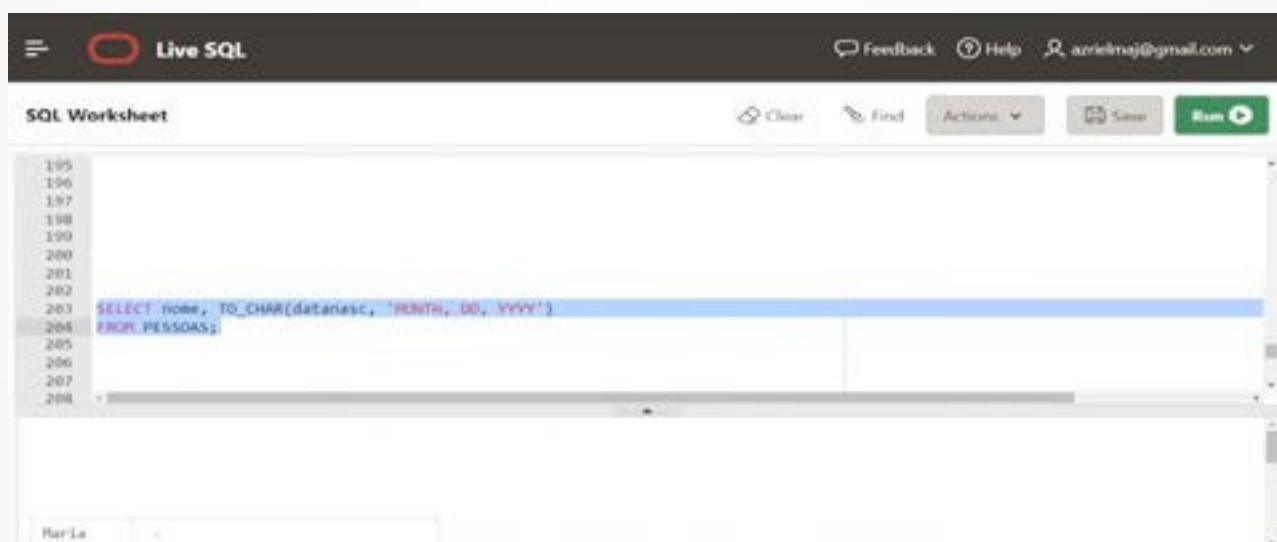
```
SELECT TO_CHAR(SYSDATE, 'MONTH, DD, YYYY HH24:MI:SS')  
FROM DUAL;
```

The query has been executed, and the result is displayed in a table below the editor:

TO_CHAR(SYSDATE, 'MONTH, DD, YYYY HH24:MI:SS')
OCTOBER , 17, 2022 17:31:22

Below the table, there is a 'Download CSV' link.

Para utilizar o comando To\_Char na tabela Pessoas, convertendo para a data no tipo mês, dia e ano, utilize o código abaixo e clique em Run:



The screenshot shows the 'Live SQL' web application. The SQL Worksheet area contains the following query:

```
SELECT nome, TO_CHAR(datanasc, 'MONTH, DD, YYYY')  
FROM PESSOAS;
```

The query is ready to be executed. The interface includes a 'Run' button in the top right corner of the worksheet area.

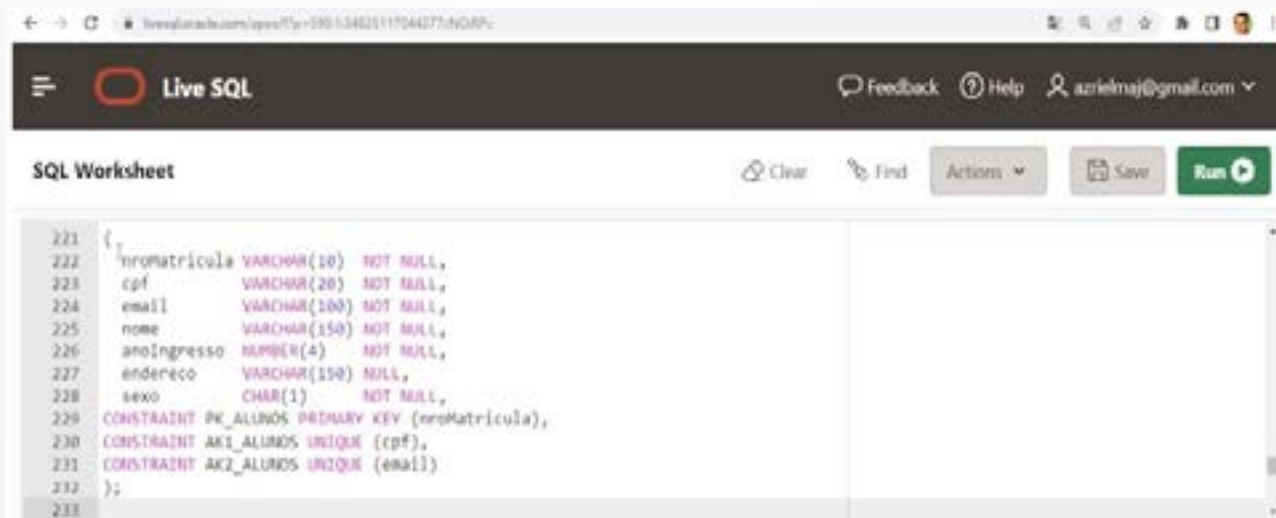
# Integridade de tabelas

A integridade dos dados refere-se à confiabilidade e consistência das informações, ao longo do seu ciclo de vida útil. Ela tem como objetivo preservar o conhecimento para que nada seja comprometido ou perdido, prejudicando, assim, todo o planejamento organizacional.

A integridade de Entidade deve ser criada com a chave primária (PK) em cada tabela. Ela não vai se repetir nunca.

Na escolha da PK, deve-se levar em consideração a chave que nunca se repete, para manter a integridade. Exemplos: CPF - cada pessoa possui apenas um CPF, logo essa pode ser uma boa escolha para uma chave primária na tabela Pessoas. Deve-se ter cuidado com possíveis alterações no formato do CPF que podem atingir a integridade da tabela. Para isso, uma boa solução é utilizar como PK a matrícula, pois, assim, teremos uma chave primária que não sofre riscos de alterações; essa chave deve ser criada sempre como Not Null. Chaves candidatas também podem ser utilizadas; são as Alternate Keys, criadas como Unique. Ao criar uma chave candidata como Unique, o Oracle não permite a inserção de dados repetidos.

Assim, crie a tabela Alunos, indicando as Constraint para a chave PK, que é a indicação de que essa é a chave primária e Constraint, utilizando Unique nas chaves alternativas:



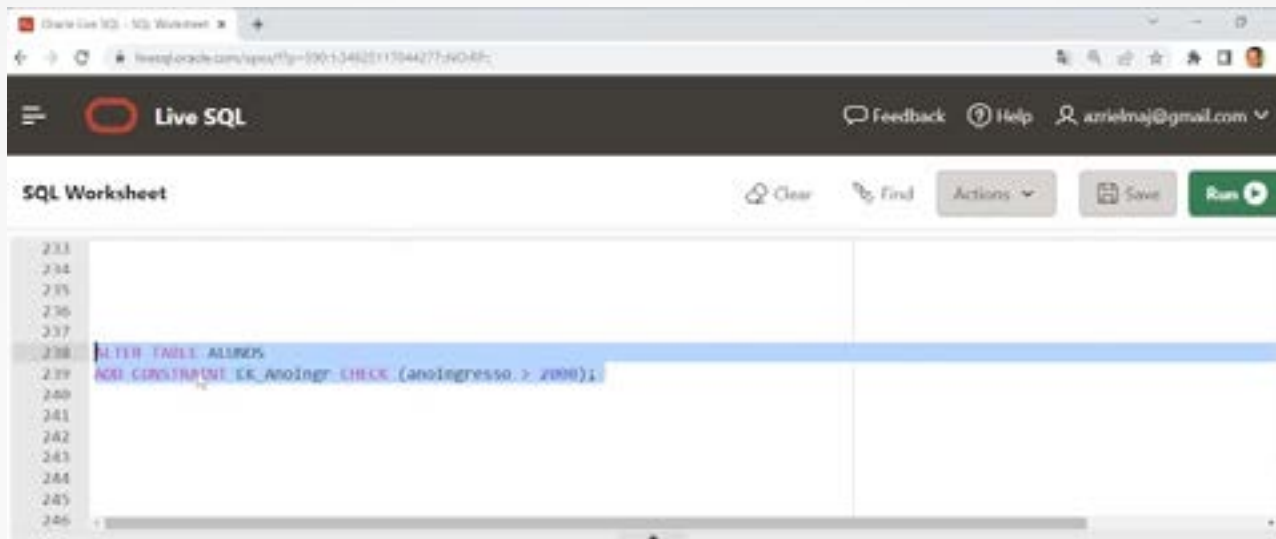
```
221 {
222   nroMatricula VARCHAR(10) NOT NULL,
223   cpf          VARCHAR(20) NOT NULL,
224   email        VARCHAR(100) NOT NULL,
225   nome         VARCHAR(150) NOT NULL,
226   anoIngresso  NUMBER(4)   NOT NULL,
227   endereco    VARCHAR(150) NULL,
228   sexo        CHAR(1)     NOT NULL,
229   CONSTRAINT PK_ALUNOS PRIMARY KEY (nroMatricula),
230   CONSTRAINT AK1_ALUNOS UNIQUE (cpf),
231   CONSTRAINT AK2_ALUNOS UNIQUE (email)
232 };
233
```

**IMPORTANTE:** na imagem, não se pode esquecer do CREATE TABLE alunos, antes dos parênteses (chaves, tipos de dados e se Not Null, com as devidas Constraints criadas). Selecione a tabela a ser criada e clique em Run.

**IMPORTANTE:** no exemplo da tabela Pessoas, observe que a PK não foi criada, o que permite repetir dados na inserção, através do Insert Into.

# Integridade de domínio

Consiste em uma diversidade de processos que asseguram a precisão de cada parte dos dados de um domínio. Ou seja, são os valores aceitáveis que cada coluna pode conter. Aqui, podem ser inseridas regras que limitam o formato e a quantidade de informações adicionadas. Para isso, usamos a restrição Check:

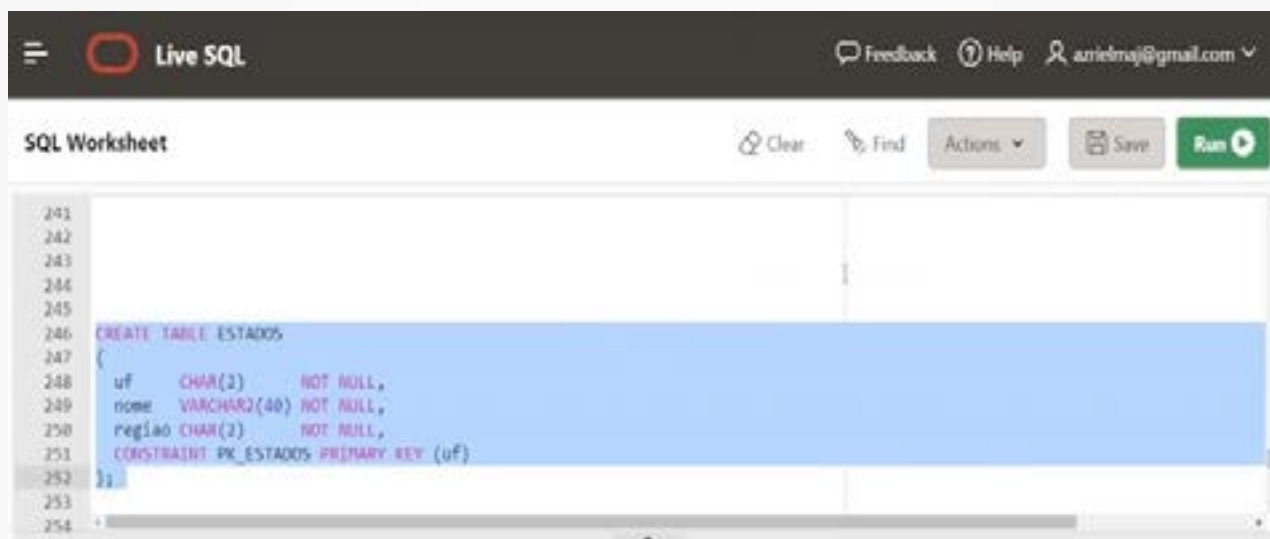


```
233  
234  
235  
236  
237  
238 ALTER TABLE ALUNOS;  
239 ADD CONSTRAINT CK_AnoIngr CHECK (anoingresso > 2000);  
240  
241  
242  
243  
244  
245  
246
```

Ao alterar a tabela alunos, está sendo adicionada uma restrição chamada CK\_AnoIngr e Check, pois o ano de ingresso deve ser maior que o ano 2000. Use o Check sempre entre parênteses.

# Integridade referencial

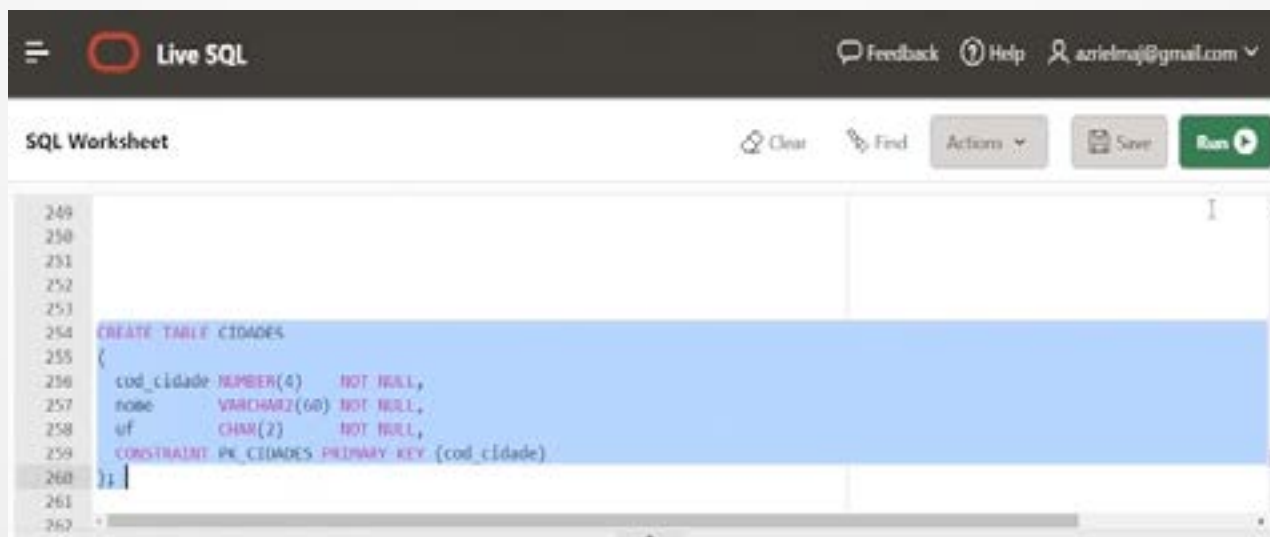
Integridade referencial é um conceito de banco de dados que garante que todos os relacionamentos propostos entre tabelas, no modelo de entidade-relacionamento (ER), serão respeitados, dando a certeza de que os dados de um banco de dados estarão íntegros, ou seja, é a criação da chave estrangeira. Veja o exemplo na criação da tabela Estados:



```
241  
242  
243  
244  
245  
246 CREATE TABLE ESTADOS  
247 (  
248   uf      CHAR(2)      NOT NULL,  
249   nome    VARCHAR2(40) NOT NULL,  
250   região  CHAR(2)      NOT NULL,  
251   CONSTRAINT PK_ESTADOS PRIMARY KEY (uf)  
252 );  
253  
254
```

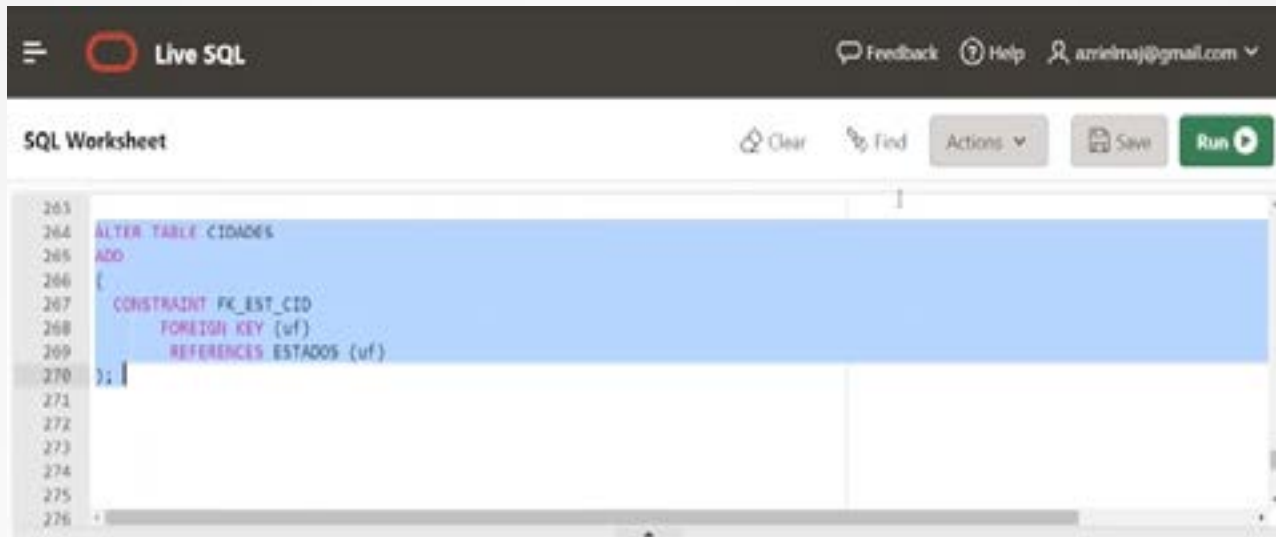
Observe que, ao criar a tabela Estados, temos a Constraint para PK\_ESTADOS e sua chave primária é uf, sempre entre parênteses.

Ao criar a tabela Cidades, como não há nenhuma restrição de integridade entre elas, é possível cadastrar cidade na coluna uf com XX, ou seja, que não existe na tabela de estados ou uma sigla qualquer que não exista.



```
249  
250  
251  
252  
253  
254 CREATE TABLE CIDADES  
255 (  
256   cod_cidade NUMBER(4) NOT NULL,  
257   nome       VARCHAR2(60) NOT NULL,  
258   uf         CHAR(2)     NOT NULL,  
259   CONSTRAINT PK_CIDADES PRIMARY KEY (cod_cidade)  
260 );  
261  
262
```

Para criar uma relação de integridade entre as tabelas Estados e Cidades, se altera a tabela Cidades e se adiciona uma Constraint ou restrição de chave estrangeira, que vai referenciar Estados. Ao rodar o comando, não será possível incluir nenhuma UF que não exista na tabela de Estados. Veja a imagem abaixo, com a relação de integridade entre as tabelas criadas:



The screenshot shows a web-based SQL editor interface. At the top, there's a dark header with a menu icon, a red circle logo, and the text "Live SQL". To the right of the header are links for "Feedback", "Help", and a user profile "azrieltraj@gmail.com". Below the header, the main area is titled "SQL Worksheet". On the right side of this area are buttons for "Clear", "Find", "Actions", "Save", and a green "Run" button with a play icon. The central part of the interface is a text editor with a light blue background. It contains the following SQL code: 

```
263  
264 ALTER TABLE CIDADES  
265 ADD  
266 {  
267     CONSTRAINT FK_EST_CID  
268     FOREIGN KEY (uf)  
269     REFERENCES ESTADOS (uf)  
270 };
```

 The code is syntax-highlighted, with keywords in purple and table names in blue. A vertical line is visible in the editor, and a scrollbar is on the right side.



# Relacionamentos

Os relacionamentos de banco de dados são associações entre tabelas que são criadas, usando instruções de junção para recuperar dados.

Relacionamentos Muitos para Muitos: no exemplo utilizado na tabela Médico e Paciente, ao serem relacionadas, verifica-se que um médico atende muitos pacientes e que um paciente é atendido por muitos médicos. Assim, o relacionamento é (N:N) em ambas tabelas. Com isso, a Chave estrangeira, ou Foreign Key (FK), deve ser usada, após a criação da tabela intermediária Atende.

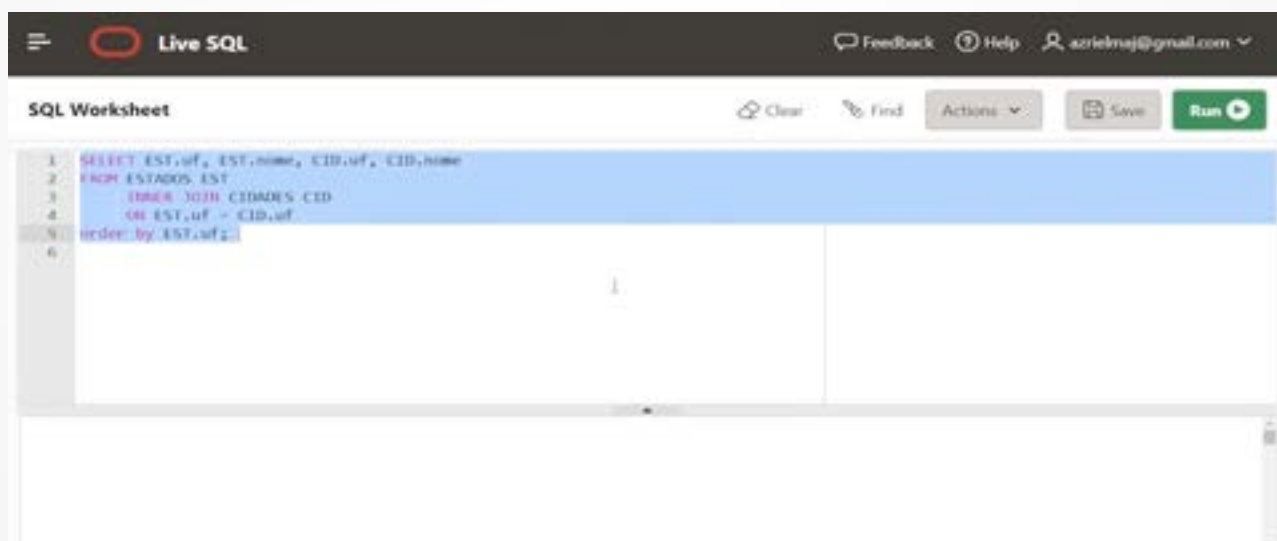
# Manipulação de dados

A manipulação de dados significa:

- buscar a informação armazenada no BD;
- inserir novas informações nos BD;
- eliminar informações no BD; e
- modificar dados armazenados no BD.

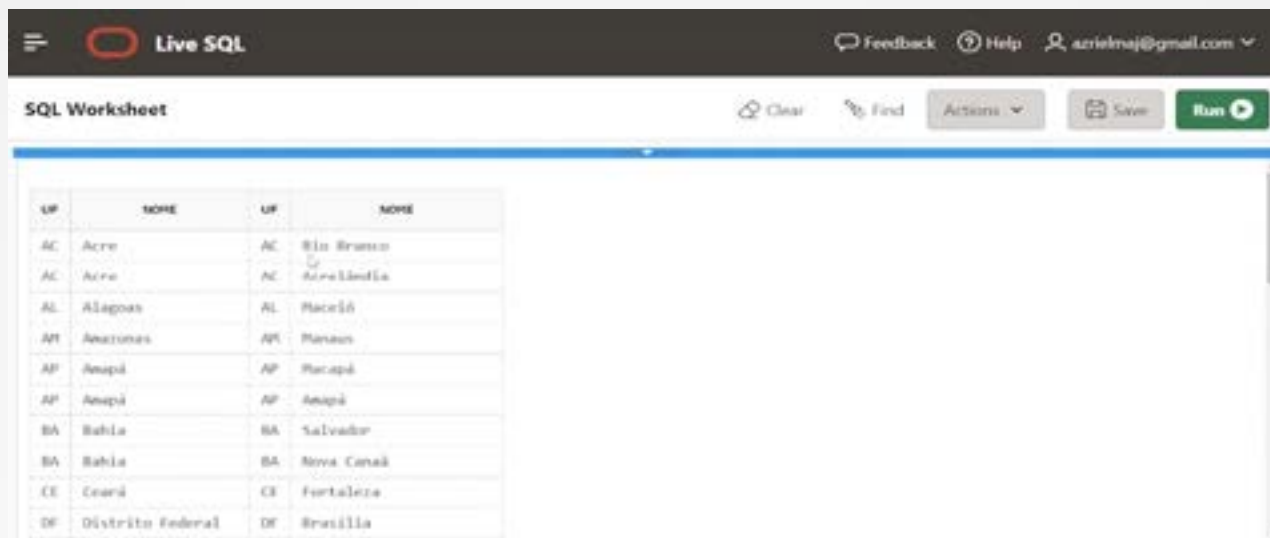
Assim, a linguagem de manipulação de dados, ou Data Manipulation Language (DML), é a linguagem que permite aos usuários fazer o acesso aos dados, ou manipulá-los, conforme modelo de dados apropriado.

O Comando JOIN permite a união de duas tabelas diferentes em uma única tabela, através de PK e FK:



O comando Inner Join junta duas tabelas, como já foi mencionado. O Select seleciona o campo da tabela que será apresentada no resultado, mas veja, em From as tabelas Estados e Cidades recebem apelidos como ESTADOS EST e CIDADES CID, apenas para facilitar a busca. Na cláusula ON, temos a condição que diz que UF da tabela Estados deve ser igual ao UF da tabela Cidades. O comando Order By pode ser usado para ordenar o campo UF da tabela Estados, em ordem ascendente.

O resultado do filtro será:



The screenshot shows the Live SQL interface with a table containing the following data:

UF	NOME	UF	NOME
AC	Acre	AC	Rio Branco
AC	Acre	AC	Acrelândia
AL	Alagoas	AL	Maceió
AM	Amazonas	AM	Manaus
AP	Paraná	AP	Paraná
AP	Paraná	AP	Paraná
BA	Bahia	BA	Salvador
BA	Bahia	BA	Nova Canaã
CE	Ceará	CE	Fortaleza
DF	Distrito Federal	DF	Brasília

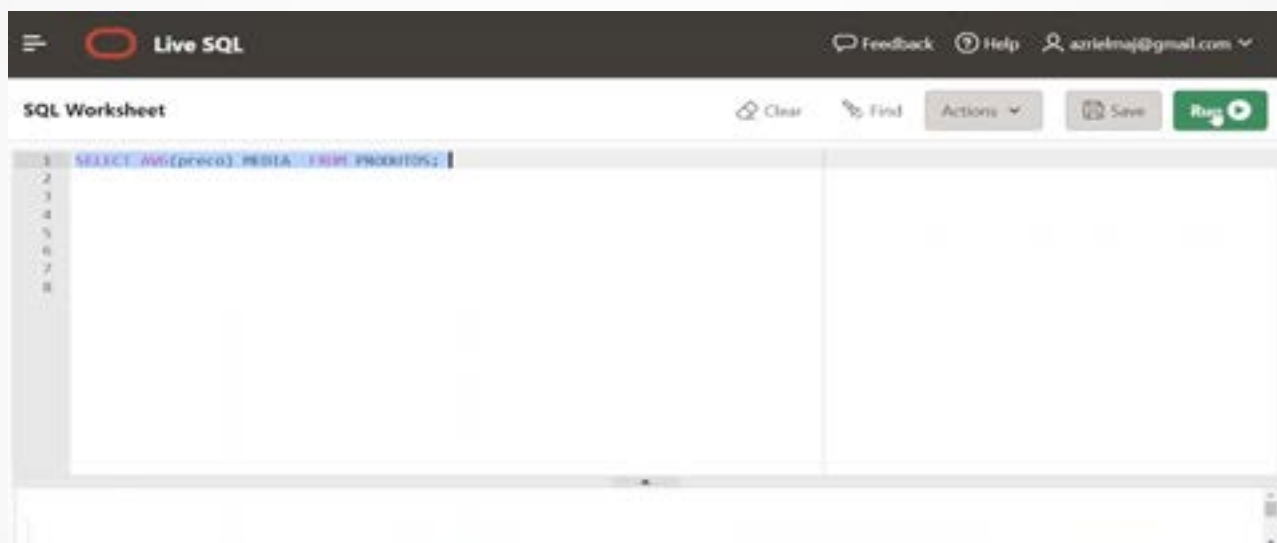
A tabela mostra justamente a UF dos Estados em ordem Ascendente e a UF das Cidades.

# Funções

Existem funções que são funções sobre linhas e funções que são sobre conjunto de linhas, para determinar totais, médias, valor, etc.

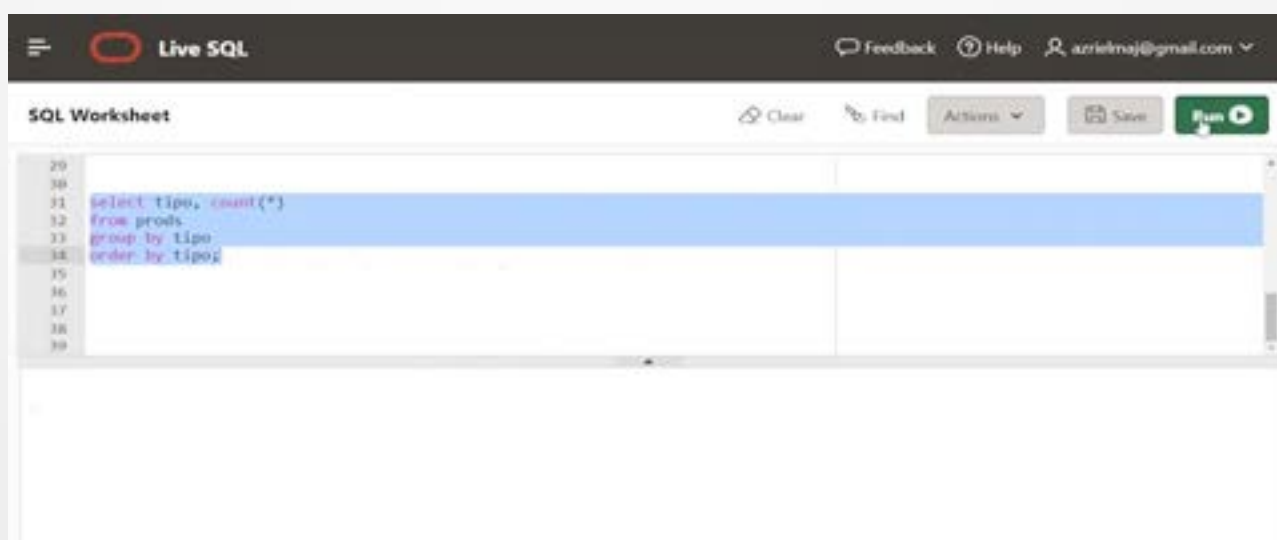
As funções de agregação operam sobre um conjunto de linhas, agrupando resultado por determinado atributo. O Count é um bom exemplo de função agregada.

Outra função de agregação usada é o AVG, que traz a média dos valores de uma determinada coluna:

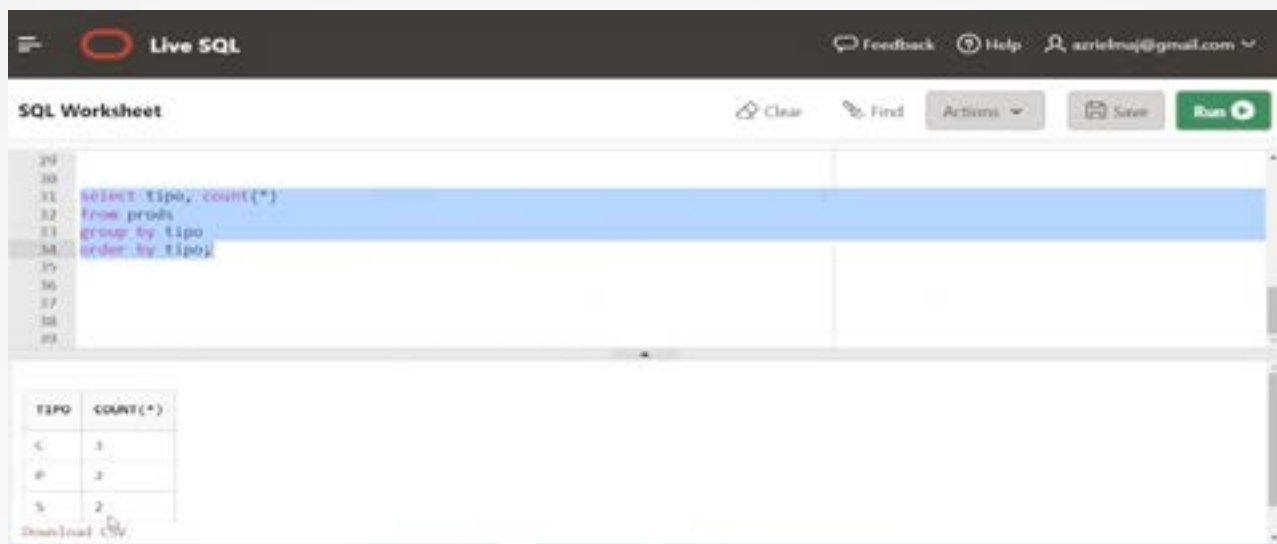


Esse Select trará a média de preços. A palavra média é um apelido para o campo onde resultado seja mostrado. O Select buscará na tabela Produtos, devido ao From.

O Group By é uma função agregadora. Esse comando também utiliza a cláusula Select para gerar subtotais em novas colunas. Veja o exemplo:



Nessa consulta, foi utilizado Count, que também é uma função agregadora, para contar o número de tipos existentes na tabela prods (essa tabela deve ser criada para testes). O Group by agrupará os resultados, e, o Order By, ordenará o resultado em ordem ascendente. Veja o resultado do filtro:



The screenshot shows the Live SQL interface. The SQL query entered is:

```
SELECT tipo, count(*)  
FROM prods  
GROUP BY tipo  
ORDER BY tipo;
```

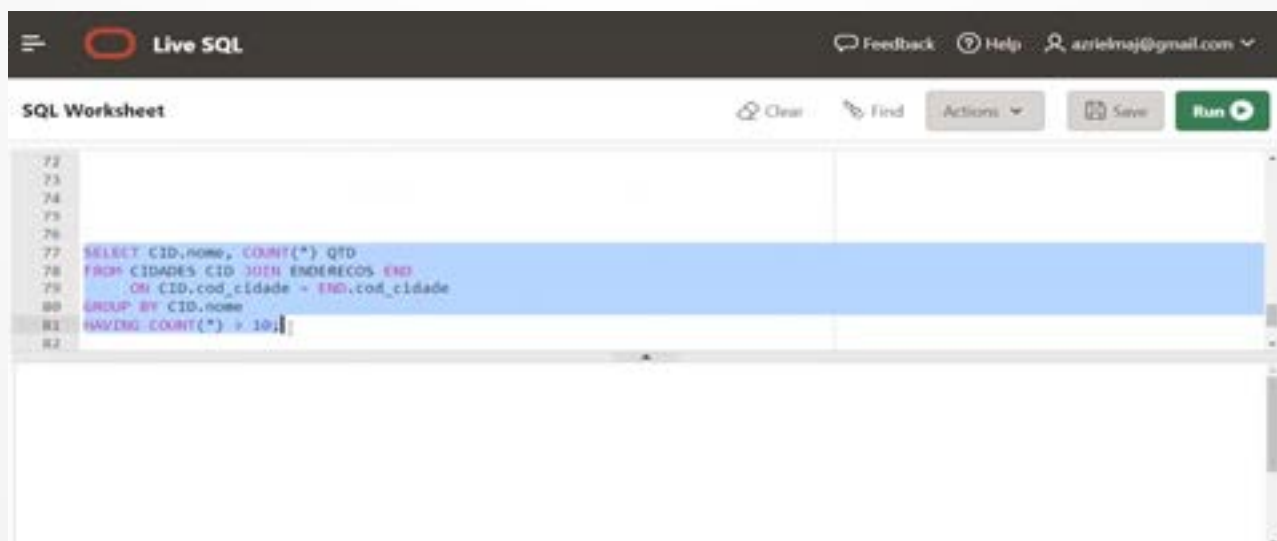
The results are displayed in a table with two columns: TIPO and COUNT(\*).

TIPO	COUNT(*)
C	3
P	2
S	2

Below the table is a "Download CSV" link.

Logo, verificamos que, o Tipo C, temos o total de 3 resultados, o Tipo P, 2 resultados, e, o Tipo S, 2 resultados.

O Having funciona como o Where do Order By. No exemplo:



The screenshot shows the Live SQL interface. The SQL query entered is:

```
SELECT CID,nome, COUNT(*) QTD  
FROM CIDADES CID JOIN ENDEREÇOS END  
ON CID.cod_cidade = END.cod_cidade  
GROUP BY CID,nome  
HAVING COUNT(*) > 10;
```

The results section is currently empty.

A consulta está selecionando a cidade, o total de cidades e endereços ordenados por cidade. Mas será apresentado apenas as cidades com número de maior de 10.

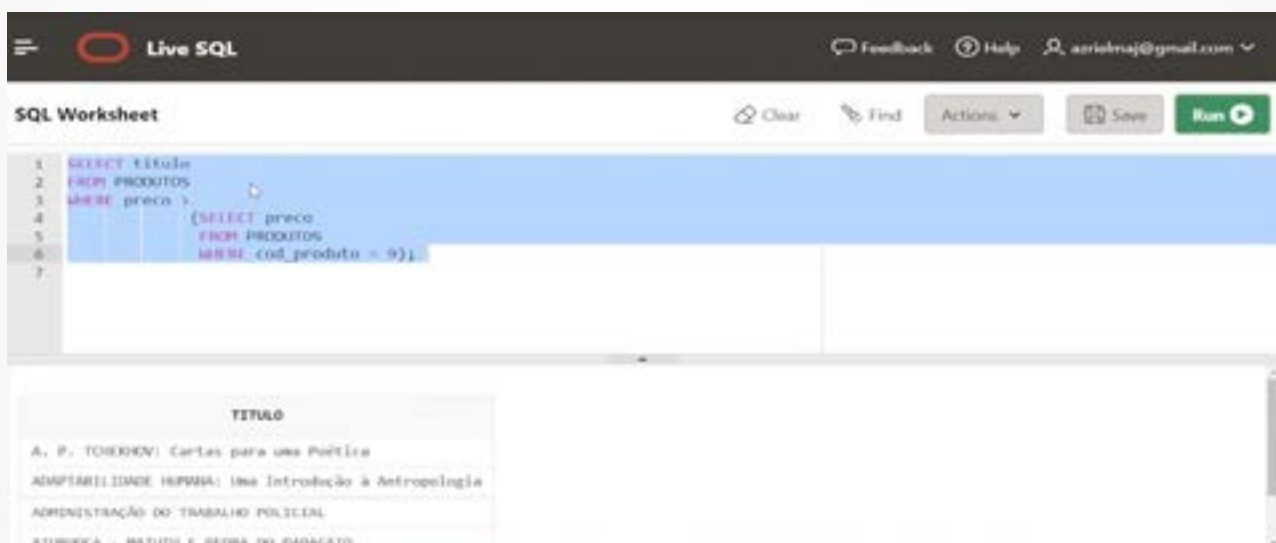
As Subconsultas permitem juntar o resultado de duas consultas em uma só:



The screenshot shows the Live SQL interface with the following SQL query:

```
1 SELECT titulo
2 FROM PRODUTOS
3 WHERE preco >
4     (SELECT preco
5      FROM PRODUTOS
6      WHERE cod_produto = 9);
```

Nessa subconsulta, a consulta principal seleciona os títulos da tabela Produtos onde preço seja maior que o resultado da subconsulta, assim:



The screenshot shows the Live SQL interface with the results of the query displayed in a table. The table has a single column titled 'TITULO' and contains four rows of data.

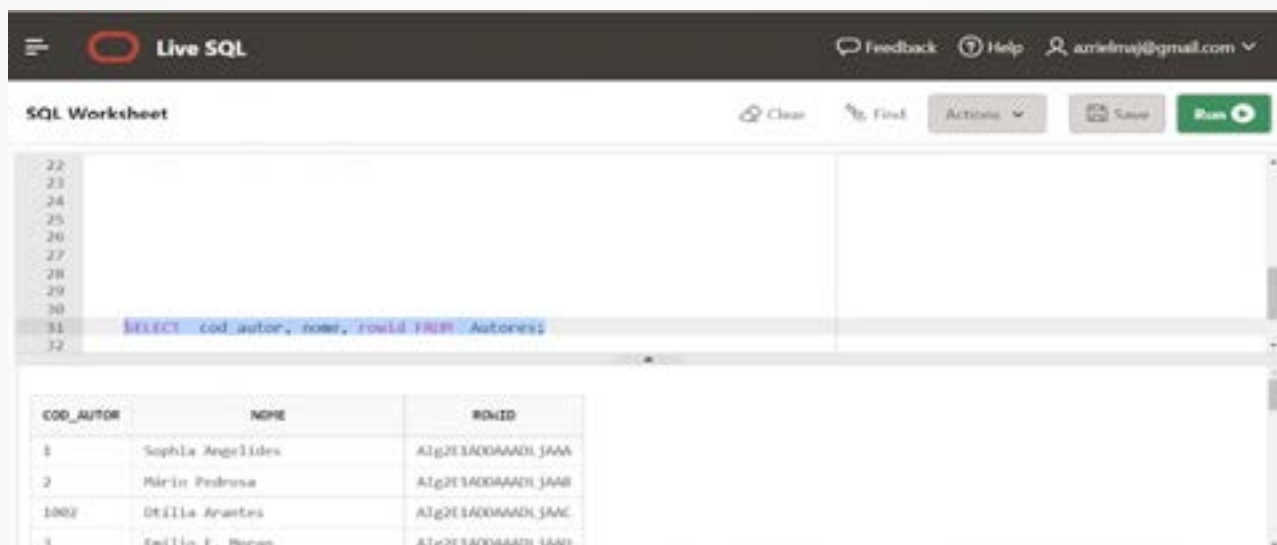
TITULO
A. P. TOLSTOY: Cartas para uma Política
ADAPTABILIDADE HUMANA: Uma Introdução à Antropologia
ADMINISTRAÇÃO DO TRABALHO POLICIAL
AJURDICA - PATUTO E PEDRA DO PARAGUAI



# Indexação

Ao selecionar cidades, elas virão ordenadas, justamente porque foram inseridas dessa forma, mas o Sistemas Gerenciadores de Banco de Dados (SGBD) não garantem a ordenação nas consultas, por isso, temos que criar o recurso que se chama Índice. O Índice nada mais é que uma tabela onde teremos separado a chave de consulta e o endereço físico no banco de dados onde queremos procurar o que se precisa.

Para isso, utilizaremos como exemplo o rowid, que é o endereço do DataBlock, onde as informações estão gravadas:

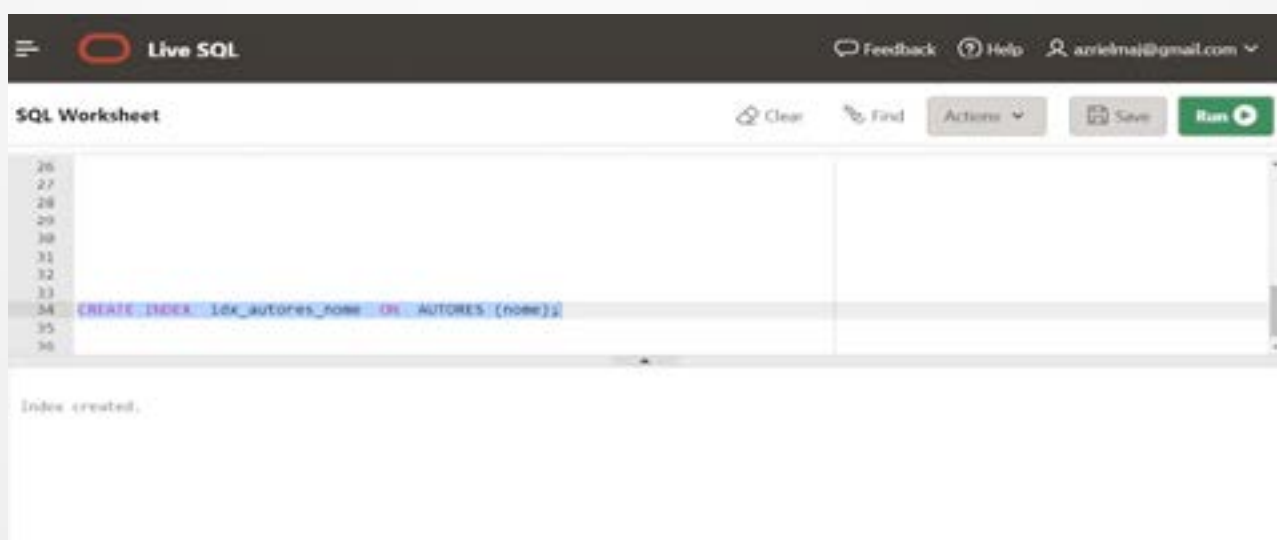


The screenshot shows a web-based SQL editor titled "Live SQL". The interface includes a top navigation bar with a menu icon, the "Live SQL" logo, and links for "Feedback", "Help", and a user profile "azrielmaj@gmail.com". Below the navigation bar is a toolbar with "Clear", "Find", "Actions", "Save", and a "Run" button. The main area is labeled "SQL Worksheet" and contains a text editor with line numbers 22 to 32. A SQL query is entered in line 31: `SELECT cod_autor, nome, rowid FROM Autores;`. Below the editor, the results of the query are displayed in a table with three columns: "COD\_AUTOR", "NOME", and "ROWID".

COD_AUTOR	NOME	ROWID
1	Sophia Angelides	A1g2E1A0DAAHJAAH
2	Márin Pedrosa	A1g2E1A0DAAHJAAH
1002	Otilia Arantes	A1g2E1A0DAAHJAAH
3	Emílio F. Moran	A1e2E1A0DAAHJAAH

A consulta trará o endereço físico de onde as informações estão gravadas.

Assim, podemos criar Índice por nome, com todos os nomes ordenados e o rowid para localizar o nome nessa tabela.



The screenshot shows the same "Live SQL" interface. The SQL Worksheet now contains a query in line 34: `CREATE INDEX idx_autores_nome ON Autores (nome);`. Below the editor, a message states "Index created,".

COD_AUTOR	NOME	ROWID
1	Sophia Angelides	A1g2E1A0DAAHJAAH
2	Márin Pedrosa	A1g2E1A0DAAHJAAH
1002	Otilia Arantes	A1g2E1A0DAAHJAAH
3	Emílio F. Moran	A1e2E1A0DAAHJAAH

O índice foi criado.

Os bancos de dados, muitas vezes, podem criar chaves sequenciais. O Oracle não faz isso, logo, deve-se criar um objeto chamado Sequence, com uma série de parâmetros, inclusive pelo número que se quer começar:



The screenshot shows the 'Live SQL' web interface. At the top, there's a navigation bar with 'Live SQL', 'Feedback', 'Help', and a user profile. Below this is a 'SQL Worksheet' section with a toolbar containing 'Clear', 'Find', 'Actions', 'Save', and 'Run'. The main text area contains the SQL command: `CREATE SEQUENCE seq_tituloacoes START WITH 6;`. Below the code, a message states 'Index created.'

Acabamos de criar uma sequência que inicia pelo número 6.

O Nextval pega o próximo valor da sequência:



The screenshot shows the 'Live SQL' web interface. The 'SQL Worksheet' section now contains the SQL command: `SELECT seq_tituloacoes.nextval FROM DUAL;`. Below the code, a table with one row is displayed, showing the value '6' under the column header 'NEXTVAL'. A 'Download CSV' link is visible below the table.

Se a consulta for executada novamente, será mostrado 7 e assim sucessivamente.

**PUCRS** online

 **uol** edtech.