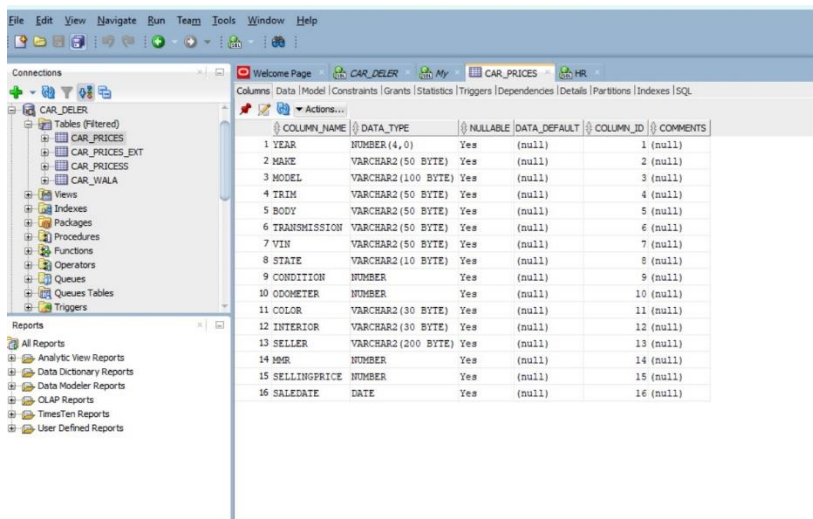


STEP1: CREATED TABLE (car_prices) :

```
CREATE TABLE CAR_PRICES (  
  
    YEAR NUMBER(4),  
  
    MAKE VARCHAR2(50),  
  
    MODEL VARCHAR2(100),  
  
    TRIM VARCHAR2(100),  
  
    BODY VARCHAR2(50),  
  
    TRANSMISSION VARCHAR2(50),  
  
    VIN VARCHAR2(17),  
  
    STATE VARCHAR2(10),  
  
    CONDITION NUMBER(3),  
  
    ODOMETER NUMBER(10),  
  
    COLOR VARCHAR2(50),  
  
    INTERIOR VARCHAR2(50),  
  
    SELLER VARCHAR2(200),  
  
    MMR NUMBER(10),  
  
    SELLINGPRICE NUMBER(10),  
  
    SALEDATE DATE  
  
);
```

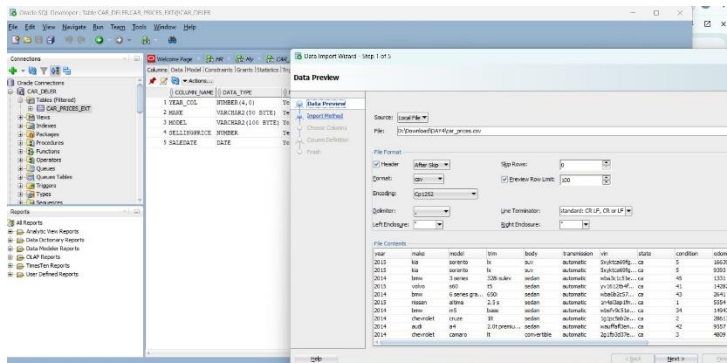


The screenshot displays the Oracle SQL Developer interface. On the left, the 'Connections' pane shows a tree view with 'CAR_DEALER' expanded, containing tables like 'CAR_PRICES', 'CAR_PRICES_EXT', 'CAR_PRICESS', and 'CAR_VIALA'. The main window shows the 'Columns' tab for the 'CAR_PRICES' table, listing 16 columns with their data types, nullability, and default values.

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 YEAR	NUMBER(4,0)	Yes	(null)	1	(null)
2 MAKE	VARCHAR2(50 BYTE)	Yes	(null)	2	(null)
3 MODEL	VARCHAR2(100 BYTE)	Yes	(null)	3	(null)
4 TRIM	VARCHAR2(50 BYTE)	Yes	(null)	4	(null)
5 BODY	VARCHAR2(50 BYTE)	Yes	(null)	5	(null)
6 TRANSMISSION	VARCHAR2(50 BYTE)	Yes	(null)	6	(null)
7 VIN	VARCHAR2(50 BYTE)	Yes	(null)	7	(null)
8 STATE	VARCHAR2(10 BYTE)	Yes	(null)	8	(null)
9 CONDITION	NUMBER	Yes	(null)	9	(null)
10 ODOMETER	NUMBER	Yes	(null)	10	(null)
11 COLOR	VARCHAR2(30 BYTE)	Yes	(null)	11	(null)
12 INTERIOR	VARCHAR2(30 BYTE)	Yes	(null)	12	(null)
13 SELLER	VARCHAR2(200 BYTE)	Yes	(null)	13	(null)
14 MMR	NUMBER	Yes	(null)	14	(null)
15 SELLINGPRICE	NUMBER	Yes	(null)	15	(null)
16 SALEDATE	DATE	Yes	(null)	16	(null)

STEP2:

I have Imported my data set car_prices.csv to Oracle DB using below commands and granted access to my SQL USER 'HR':



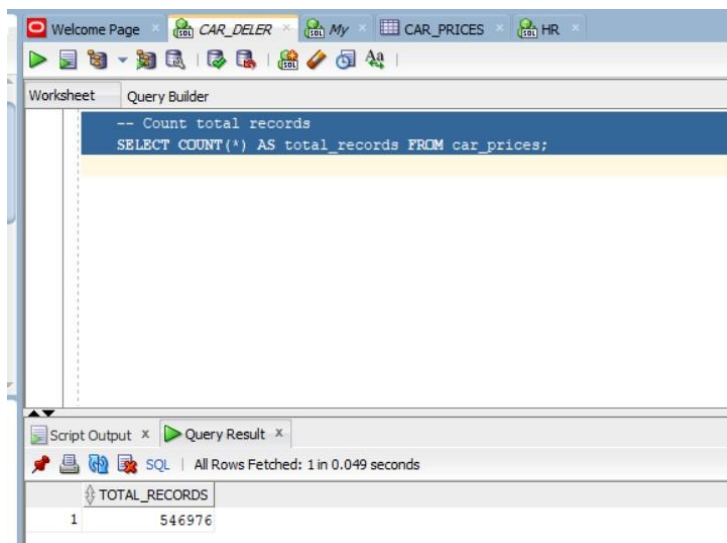
```
CREATE OR REPLACE DIRECTORY car_dir AS 'D:\Download\DAY4\car_prices.csv';
```

```
GRANT READ, WRITE ON DIRECTORY car_dir TO CAR_DELER
```

STEP3:

Basic analysis query's used:

```
SELECT COUNT(*) AS total_records FROM car_prices;
```



SELECT * FROM car_prices WHERE ROWNUM <= 10;

	YEAR	MAKE	MODEL	TRIM	BODY	TRANSMISSION	VIN	STATE	CONDITION	ODOMETER	COLOR	INTERIOR	SELLER
1	2013	hyundai	elantra	gl	sedan	automatic	5npd4ae7dh327127	ca	2	29006	gray	gray	enterprise vehicle
2	2013	hyundai	elantra	gl	sedan	automatic	kmhd4ae7da946905	ca	48	8555	red	beige	hyundai buybacks
3	2013	infiniti	g sedan	g37	journey g sedan	automatic	jnlcv6ap1dm712273	ca	39	26907	gray	black	nissan infiniti lt
4	2013	infiniti	g sedan	g37	journey g sedan	automatic	jnlcv6ap5dm712977	ca	34	14765	gray	black	nissan infiniti lt
5	2013	infiniti	g sedan	g37	journey g sedan	automatic	jnlcv6ap3dm710802	ca	43	18673	gray	gray	nissan infiniti lt
6	2013	infiniti	g sedan	g37	journey g sedan	automatic	jnlcv6ap2dm712489	ca	37	20707	white	black	nissan infiniti lt
7	2013	hyundai	veloster	turbo	hatchback	automatic	kmhtc6ae3dul46243	ca	42	8467	black	black	hyundai motor amer
8	2013	infiniti	g sedan	g37	journey g sedan	automatic	jnlcv6ap4dm715250	ca	44	16009	gray	black	nissan infiniti lt
9	2013	infiniti	g sedan	g37	journey g sedan	automatic	jnlcv6ap0dm714516	ca	4	15260	silver	black	nissan infiniti lt
10	2013	hyundai	sonata hybrid base		sedan	automatic	kmhec44kda091084	ca	44	9539	silver	gray	hyundai motor amer

SELECT

COUNT(*) - COUNT(year) AS missing_year,

COUNT(*) - COUNT(make) AS missing_make,

COUNT(*) - COUNT(sellingprice) AS missing_price,

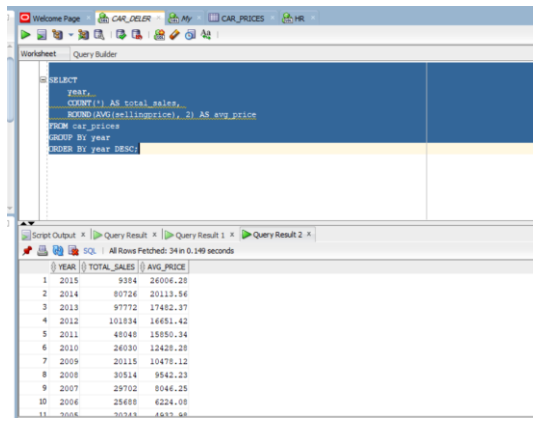
COUNT(*) - COUNT(mmr) AS missing_mmr

FROM car_prices;

	MISSING_YEAR	MISSING_MAKE	MISSING_PRICE	MISSING_MMR
1	0	0	0	0

STEP4: Sales Analysis by Year

```
SELECT
    year,
    COUNT(*) AS total_sales,
    ROUND(AVG(sellingprice), 2) AS avg_price
FROM car_prices
GROUP BY year
ORDER BY year DESC;
```



The screenshot shows a SQL query editor with the following SQL code:

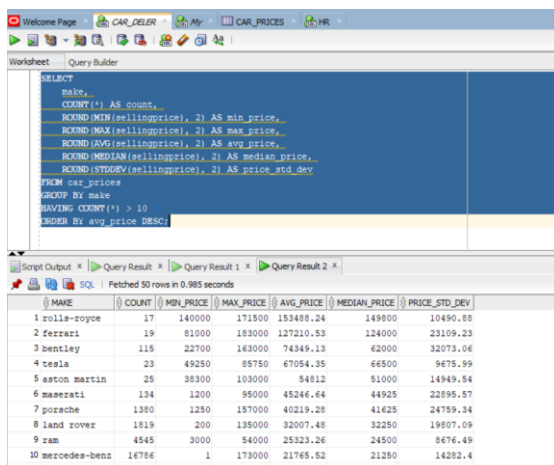
```
SELECT
    year,
    COUNT(*) AS total_sales,
    ROUND(AVG(sellingprice), 2) AS avg_price
FROM car_prices
GROUP BY year
ORDER BY year DESC;
```

The results are displayed in a table with 11 rows and 3 columns: YEAR, TOTAL_SALES, and AVG_PRICE.

YEAR	TOTAL_SALES	AVG_PRICE
2015	9394	26006.28
2014	89726	20113.86
2013	97772	17482.37
2012	101834	16651.42
2011	48048	15850.34
2010	26030	12428.28
2009	20115	10476.12
2008	20514	9542.23
2007	29702	8046.25
2006	25688	6224.08
2005	30343	4833.88

STEP5: Price Analysis By BRAND

```
SELECT
    make,
    COUNT(*) AS count,
    ROUND(MIN(sellingprice), 2) AS min_price,
    ROUND(MAX(sellingprice), 2) AS max_price,
    ROUND(AVG(sellingprice), 2) AS avg_price,
    ROUND(MEDIAN(sellingprice), 2) AS median_price,
    ROUND(STDDEV(sellingprice), 2) AS price_std_dev
FROM car_prices
GROUP BY make
HAVING COUNT(*) > 10
ORDER BY avg_price DESC;
```



The screenshot shows a SQL query editor with the following SQL code:

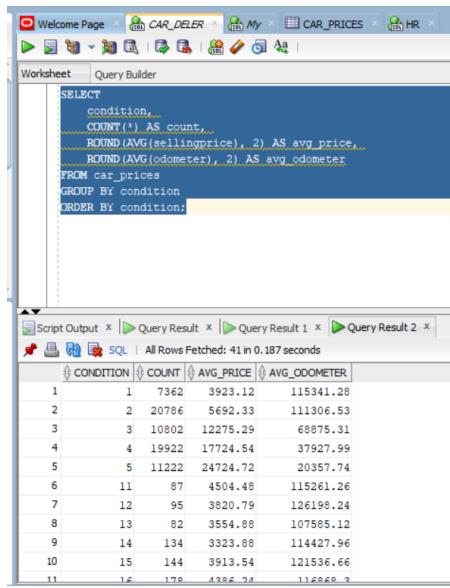
```
SELECT
    make,
    COUNT(*) AS count,
    ROUND(MIN(sellingprice), 2) AS min_price,
    ROUND(MAX(sellingprice), 2) AS max_price,
    ROUND(AVG(sellingprice), 2) AS avg_price,
    ROUND(MEDIAN(sellingprice), 2) AS median_price,
    ROUND(STDDEV(sellingprice), 2) AS price_std_dev
FROM car_prices
GROUP BY make
HAVING COUNT(*) > 10
ORDER BY avg_price DESC;
```

The results are displayed in a table with 11 rows and 7 columns: MAKE, COUNT, MIN_PRICE, MAX_PRICE, AVG_PRICE, MEDIAN_PRICE, and PRICE_STD_DEV.

MAKE	COUNT	MIN_PRICE	MAX_PRICE	AVG_PRICE	MEDIAN_PRICE	PRICE_STD_DEV
rolls-royce	17	140000	171500	153485.24	149500	10490.88
ferarri	19	51000	183000	127210.53	124000	23109.23
bentley	115	22700	163000	74349.13	62000	32073.06
tesla	23	46250	65750	67054.35	66500	9676.99
aston martin	25	38300	103000	54812	51000	14949.54
maserati	134	1200	95000	45246.64	44925	22896.57
porcche	1380	1250	157000	40219.28	41625	24759.34
land rover	1819	200	135000	32007.48	32250	19807.09
ram	4545	3000	54000	25323.26	24500	8676.49
mercedes-benz	16786	1	173000	21765.52	21250	14282.4
bmw	26578	100	128000	21753.16	20600	14680.8

STEP6: Vehicle Condition Analysis (Average price by condition rating)

```
SELECT
    condition,
    COUNT(*) AS count,
    ROUND(AVG(sellingprice), 2) AS avg_price,
    ROUND(AVG(odometer), 2) AS avg_odometer
FROM car_prices
GROUP BY condition
ORDER BY condition;
```



	CONDITION	COUNT	AVG_PRICE	AVG_ODOMETER
1	1	7362	3923.12	115341.28
2	2	20786	5692.33	111306.53
3	3	10802	12275.29	68875.31
4	4	19922	17724.54	37927.99
5	5	11222	24724.72	20357.74
6	11	87	4504.48	115261.26
7	12	95	3820.79	126198.24
8	13	82	3554.88	107585.12
9	14	134	3323.88	114427.96
10	15	144	3913.54	121536.66
11	16	178	4386.24	116869.3

Analyze cars with body type classification

```
WITH body_categories AS (
    SELECT 'SEDAN' AS body_type FROM DUAL
    UNION ALL SELECT 'SUV' FROM DUAL
    UNION ALL SELECT 'COUPE' FROM DUAL
    UNION ALL SELECT 'CONVERTIBLE' FROM DUAL
    UNION ALL SELECT 'WAGON' FROM DUAL
    UNION ALL SELECT 'HATCHBACK' FROM DUAL
)
SELECT
    bc.body_type,
    COUNT(cp.vin) AS vehicle_count,
    ROUND(AVG(cp.sellingprice), 2) AS avg_price,
    ROUND(AVG(cp.odometer), 2) AS avg_mileage
FROM body_categories bc
LEFT JOIN car_prices cp ON UPPER(cp.body) = bc.body_type
GROUP BY bc.body_type
ORDER BY vehicle_count DESC;
```

Welcome Page x CAR_DELER x My x CAR_PRICES x HR x

Worksheet Query Builder

```
UNION ALL SELECT 'CONVERTIBLE' FROM DUAL
UNION ALL SELECT 'WAGON' FROM DUAL
UNION ALL SELECT 'HATCHBACK' FROM DUAL
)
SELECT
    bc.body_type,
    COUNT(cp.vin) AS vehicle_count,
    ROUND(AVG(cp.sellingprice), 2) AS avg_price,
    ROUND(AVG(cp.odometer), 2) AS avg_mileage
FROM body_categories bc
LEFT JOIN car_prices cp ON UPPER(cp.body) = bc.body_type
GROUP BY bc.body_type
ORDER BY vehicle_count DESC;
```

Script Output x Query Result x Query Result 1 x Query Result 2 x Query Result 3 x

SQL All Rows Fetched: 6 in 0.239 seconds

	BODY_TYPE	VEHICLE_COUNT	AVG_PRICE	AVG_MILEAGE
1	SEDAN	236847	11841.12	62298.62
2	SUV	140655	16290.92	71353.05
3	HATCHBACK	25769	10131.5	51342.72
4	COUPE	17288	16192.16	66572.58
5	WAGON	15693	10271.26	73753.57
6	CONVERTIBLE	10227	18100.23	60657.51